Universidad Politécnica de Madrid

Facultad de Informática

Departamento de Inteligencia Artificial

TESIS DOCTORAL

ABC^2: Un Modelo para el Control de Robots Autónomos

Autor: Vicente Matellán Olivera

Director: Dr. Daniel Borrajo Millán

Tesis Doctoral: ABC^2: Un Modelo para el Control de Robots Autónomos

Autor: Vicente Matellán Olivera, Licenciado en Informática

Director: Daniel Borrajo Millán, Doctor en Informática

El tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid el día 1 de Junio de 1998.

Presidente:

Vocal:

Vocal:

Vocal:

Secretario:

Realizado el acto de defensa y lectura de la Tesis el día de de 1998 en la Facultad de Informática, Boadilla del Monte (Madrid), acuerda otorgarle la calificación de

El Presidente Los Vocales

El Secretario

A mi padre

Agradecimientos

Es una tradición finalizar la escritura de una tesis doctoral expresando el agradecimiento hacia todos aquellos que la hicieron posible. Cumplo con ella con gusto, comenzando por mi tutor, por su tiempo, por haber compartido conmigo su visión de la investigación y del trabajo en la universidad y por muchas otras cosas fundamentales para terminar esta tesis.

Mi reconocimiento para todos aquellos que han compartido conmigo parte de este esfuerzo, para mis compañeros del difunto Laboratorio de Agentes Inteligentes (LAI). Juntos tuvimos algunas ideas, las discutimos y las probamos. También para los alumnos que trabajaron en el LAI y para los que hicieron posible que existiera. Agradecimiento especial a mis amigos del Grupo de Sistemas y Comunicaciones (GSyC), por aguantar estoicamente mis tribulaciones durante nuestros cafés y por hacer sugerencias desde dominios diferentes al de esta tesis.

Ampliando el círculo están los que han prestado los apoyos materiales siempre necesarios, el Departamento de Informática de la Universidad Carlos III, donde he trabajado los últimos cinco años; el grupo de gente que me acogió en Carnegie y compartió conmigo su enorme experiencia, a los organizadores de la RoboCup'97 por financiarme la asistencia a Nagoya y a los creadores del simulador de la RoboCup, enfrente del cual he pasado muchas horas, por su gran trabajo.

Restan aquellos que están más próximos: a mi familia, por sus sacrificios para mi bienestar; a mis amigos de León, por seguir ahí a pesar de las ausencias; a los de Madrid por soportar el "único" tema de conversación.

Por último, el más profundo y sentido de los reconocimiento para Camino, a la que se podrían aplicar la mayoría de los agradecimientos anteriores pero que además ha sido el empuje, el apoyo y el consuelo de muchos años; en fin, a ella, por todo.

Resumen

El objetivo de esta tesis doctoral es el diseño de un modelo para el control de robots autónomos. Su prinicipal aportación es la organización de la ejecución de tareas de alto nivel de forma oportunista, haciendo uso de controladores de bajo nivel. El modelo tiene también en consideración la posible existencia de otros robots en el mismo dominio, haciendo que sea posible la cooperación entre varios de ellos para la realización de tareas que se puedan beneficiar de la colaboración.

La principal aportación de este trabajo es el diseño del mecanismo de control de las actividades de un robot. Dicho mecanismo está basado en la utilización de una agenda para la planificación de las acciones. Los elementos que se pueden insertar en esa agenda se denominan *actos* y representan acciones potenciales del robot. La generación de estos actos puede deberse a metas definidas para el robot, a peticiones de otros robots, o a eventos generados en el entorno. La política de gestión de la agenda está predefinida, pero puede modificarse mediante la definición de heurísticas de control.

El modelo prevé también la integración entre las tareas a desarrollar individualmente en un robot y aquéllas que implican una colaboración entre robots. Para ello se ha diseñado un mecanismo de definición de las *habilidades* particulares de cada robot y otro de comunicación de las mismas a otros robots.

El mecanismo de definición de habilidades permite la utilización de distintos paradigmas para su diseño, como la lógica borrosa, las redes neuronales, o el empleo de técnicas de aprendizaje automático. Este formalismo de definición proporciona un nivel de abstracción sobre su implementación que permite la difusión de la información sobre las habilidades de un robot entre distintos robots.

Para comprobar la validez del modelo desarrollado se han realizado diversas aplicaciones, tanto con robots simulados como con robots reales. En la presente memoria se presentan y discuten las decisiones tomadas en el diseño del modelo y los resultados obtenidos en los experimentos realizados.

Abstract

The goal of this thesis is the design of a model for the control of autonomous robots. This model should allow the execution of high level tasks, using low level controllers. The existence of other robots in the same domain has also been considered. The model should provide mechanisms to let robots cooperate in tasks that require collaboration.

The main contribution of this work is the design of the control mechanism of the robot tasks. This mechanism is based on the use of an agenda as the base for the planning of the robot activities. The items that can be inserted into the agenda are named acts. These acts represent potential actions that the robot can accomplish. The acts can be generated by the internal goals of the robot, by requests from other robots, or by events generated in the environment. The administration of the agenda is predefined, but it can be modified by the definition of control heuristic.

The model also considers the integration of individual tasks and those that need collaboration with other robots. A mechanism for defining the particular skills of each robot has been defined, as well as a communication mechanism, in order to let the robot cooperate.

The skills definition mechanism allows the use of different paradigms in its design, as for instance, fuzzy logic, neural networks or the use of machine learning techniques. This definition formalism provides an abstraction level over the implementation of the skills that allows the broadcasting of the information about the individual skills among different robots.

In order to test the validation of the model, different applications have been developed. These applications involved simulated robots as well as real robots. This manuscript presents and discusses the decisions taken in the design of the model as well as the results obtained in the experiments.

Índice General

Índice de Figuras

Índice de Tablas

Formalizaciones

Capítulo 1

Introducción

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning.", the King said, very gravely,
"and go on till you come to the end: then stop."
Lewis Carroll

En este primer capítulo se describe el contexto general en el que se enmarca la presente tesis doctoral, enumerando las diferentes áreas de conocimiento que han tenido alguna influencia sobre ella y con las que se relaciona. Se exponen también las motivaciones y objetivos de la misma, finalizando con una descripción de la organización del manuscrito.

Así, en la primera sección se define brevemente el término robótica. A continuación, se introduce el campo de la inteligencia artificial, haciendo especial hincapié en su interacción con la robótica. Seguidamente se analizan otras áreas de la informática que también se han tenido en cuenta a la hora de diseñar el modelo que se propone, como son los sistemas multiagente, los sistemas tolerantes a fallos, la lógica borrosa y las técnicas de aprendizaje automático. Por último, se enumeran diversos campos no relacionados directamente con la informática pero de los que se han tomado conceptos, ideas o términos que se utilizan a lo largo de este manuscrito. Así se describen aportaciones de la psicología, la sociología y la etología.

En la segunda sección, se analizan las motivaciones del presente trabajo y los objetivos que se persiguen con su desarrollo, justificando el interés de la investigación en este campo concreto. Por último, se resumen brevemente los contenidos y la estructura de la presente memoria, justificando la organización elegida e indicando qué se puede encontrar en cada uno de sus capítulos.

1.1 Contexto

En la presente memoria se describe un modelo para el control de agentes autónomos, denominado ABC^2(**A**genda **B**ased **C**ontrol for **A**gent's **B**ehaviors **C**oordination), o lo que es igual, Control Basado en Agenda para la Coordinación de Comportamientos de Agentes. La mayoría de los experimentos realizados para demostrar la validez del modelo propuesto se han realizado en el dominio de la robótica (los detalles se analizan en los Capítulos 5 y 6), por lo que el título de la memoria hace referencia al control de robots autónomos. Sin embargo, el modelo tiene carácter genérico, sirviendo tanto para el control de robots como de agentes software.

El propio nombre del modelo intenta resumir sus fundamentos. Sin embargo, muchos de los términos que lo forman tienen diversas interpretaciones según el dominio en el que se empleen. Por ello, esta primera sección se dedica a presentar brevemente las áreas de conocimiento que han influido en su diseño.

1.1.1 Robótica

La palabra *robot* deriva de la palabra checa *robota*, que significaba trabajo. El autor dramático checo Karel Capek fue quien primero la utilizó, en la sátira teatral R.U.R. [Capek, 1921], para describir las máquinas diseñadas para sustituir al ser humano en determinados trabajos repetitivos. Por su parte, la Real Academia Española de la Lengua define *robótica* como la "técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos [...]".

Este es el entorno básico en el que se asienta el presente trabajo, la aplicación de técnicas informáticas al campo de la robótica. Dentro del amplio dominio de la informática, en esta tesis se han empleado fundamentalmente técnicas y métodos del área de la Inteligencia Artificial.

1.1.2 Inteligencia Artificial

Se suele considerar que el término Inteligencia Artificial (en adelante IA) lo acuñó John McCarthy en 1956, durante la conferencia celebrada en el Darmouth College (EE.UU.). En esta conferencia se reunieron los fundadores de las dos ramas básicas que han estructurado la IA hasta nuestros días. Por una parte, Allen Newell y Herbert Simon establecieron en la Universidad de Carnegie Mellon algunos de los principios de lo que hoy se conoce como *IA Simbólica*. Por otra parte, la escuela fundada en el

M.I.T. por Marvin Minsky y John McCarthy se ha centrado fundamentalmente en la denominada *IA Subsimbólica.*

Simplificando, se puede resumir la IA simbólica como el diseño de sistemas, generalmente basados en modelos antropomórficos, cuyo comportamiento pueda considerarse como inteligente. Su trabajo se ha centrado mayoritariamente en el diseño de estrategias de búsqueda de soluciones sobre el conjunto, generalmente muy extenso, de soluciones posibles.

En el caso del control de robots, el empleo de la IA simbólica como estrategia de control implica, por lo general, que el sistema que controla el robot se basa en la creación de un mapa interno del mundo, representado generalmente de forma lingüística. Las decisiones sobre las acciones del robot se toman realizando un análisis de los efectos que tendrían las acciones del robot, para lo cual se emplea el modelo del mundo y una representación de las acciones del robot y sus efectos.

El planteamiento de la IA subsimbólica es diferente. Sostiene que es posible diseñar sistemas inteligentes sin necesidad de mantener una representación interna del mundo externo. En una frase de Rodney Brooks "El mundo es su mejor modelo" [Brooks, 1991b]. Estas arquitecturas, conocidas también como reactivas, se basan en utilizar la información obtenida del entorno para la toma de decisiones inmediatas. Para ello utilizan distintos procesos simples, cada uno con su propio bucle de percepción-reacción. Estos procesos están basados en distintas técnicas de la IA como las redes neuronales, las máquinas de estado finito, etc.

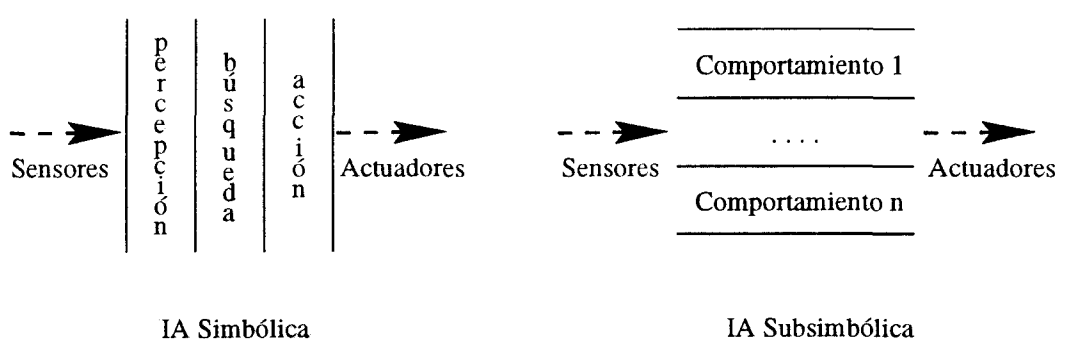

Figura 1.1: Escuelas de Inteligencia Artificial.

La Figura 1.1 resume gráficamente la diferencia entre las dos escuelas. La primera ve las acciones como el resultado de un proceso basado en pasos sucesivos: primero se procesa la información, es decir, se genera un modelo interno del mudo exterior; a continuación se realiza un proceso de búsqueda de la solución en ese modelo interno;

por último, las acciones que en el modelo interno parecían llevar a la solución se aplican en el externo. La segunda ve cada acción potencial como un enlace directo entre las entradas y las salidas del sistema.

El desarrollo de ABC^2 ha sido influido por ambas corrientes. Por una parte, se ha buscado que los robots desarrollen su trabajo en entornos no estructurados, donde la capacidad de reacción (reactividad) debe ser elevada. Por ello se ha diseñado un modelo en el que las acciones simples pueden entenderse como comportamientos puramente reactivos. Por otra parte, se deseaba que los robots pudiesen enfrentarse a tareas de mayor complejidad que las puramente reactivas, por lo que se han empleado conceptos de la IA simbólica para combinar dichos comportamientos básicos de forma adecuada.

Existen diferentes aproximaciones para conseguir este objetivo. En concreto ABC^2 se puede enmarcar dentro de los sistemas denominados oportunistas. Este tipo de aproximación, híbrido entre sistema reactivo y planificado, recibe su nombre de la capacidad para adaptar sus acciones a los eventos aleatorios que se producen en su entorno, manteniendo una batería de posibles acciones a realizar, de entre las que se escoge la que se realizará según la información percibida del entorno.

Además de por su capacidad reactiva, se ha elegido la aproximación oportunista por la facilidad que aporta a la hora de integrar en el modelo las acciones de los otros agentes. Es decir, la facilidad para abordar la parte de cooperación. La siguiente sección introduce brevente este subcampo de la IA, denominado IA Distribuida.

1.1.3 Inteligencia Artificial Distribuida

Durante los últimos años se ha incrementado el interés por el diseño de sistemas compuestos por múltiples robots autónomos. A ello ha contribuido por una parte la aparición de plataformas autónomas de bajo coste, con sistemas de control basados en micro-procesadores de propósito general, y por otra el interés por los sistemas distribuidos en general, que tiene su reflejo en todas las áreas de la informática: bases de datos, sistemas operativos, etc. En general los sistemas distribuidos proporcionan un aumento de las prestaciones, mayor tolerancia a fallos, mejor capacidad de análisis del comportamiento, etc.

El diseño de ABC^2 se ha realizado considerando la cooperación entre los agentes como uno de sus objetivos. Por ello el sistema puede definirse como un sistema para el control de agentes cooperantes, especialmente probado para el caso de agentes físicos o robots.

El estudio de los sistemas formados por varios robots extiende de forma natural la investigación en robots individuales y por consiguiente parte de sus mismos principios. En general se puede decir que existen dos aproximaciones básicas a la hora de abordar este problema: la *cooperación intencional* y los comportamientos *colectivos.*

La primera supone que los robots tienen un alto grado de inteligencia, es decir, pueden realizar tareas significativas individualmente. La cooperación se produce porque ésta ayuda a cada robot a realizar su misión [Premvuti and Yuta, 1990]. Dicha cooperación se realiza mediante una división, asignación o compartición de tareas entre los distintos robots realizada mediante comunicación entre ellos. El estudio de este tipos de sistemas se engloba dentro del campo de la inteligencia artificial distribuida [Bond and Gasser, 1988] denominado *sistemas multiagente.*

La segunda aproximación estudia el comportamiento de grupos de robots, generalmente homogéneos, que individualmente tienen pocas capacidades, pero que cuando se combinan con otros pueden generar comportamientos colectivos más útiles. Esta aproximación se basa generalmente en controlar cada robot mediante comportamientos reactivos, evitando el uso de modelos del mundo y la comunicación simbólica [Arkin, 1992]. Utilizan, en cambio, la interacción extensiva con el mundo real para producir el resultado global deseado.

La ventaja que aporta esta solución distribuida es evitar el cuello de botella presente en controladores centralizados, reduciendo la susceptibilidad del equipo a los fallos en robots individuales e incrementando su reactividad a un entorno dinámico, aunque con el coste de incrementar la dificultad en mantener la coherencia global. El estudio de este tipo de sistemas se suele englobar bajo la denominación de comportamientos emergentes, puesto que el efecto global surge como efecto lateral de la ejecución de comportamientos independientes. El problema más difícil es predecir el comportamiento global de los sistemas emergentes a partir del diseño de las reglas de control individuales de cada robot. La metodología típica es proponer una regla de control local, que permita a un grupo de robots resolver un problema dado, y a continuación estudiar el comportamiento resultante del grupo utilizando bien una simulación, o bien robots reales. Esta metodología suele basarse en resultados matemáticos de convergencia.

El diseño de ABC^2, aunque es un modelo centrado en el control y planificación de las tareas de un robot, ha tenido muy presente la posibilidad de ser utilizado en diferentes robots, proporcionando los mecanismos básicos de cooperación. Para ello se ha definido un sistema de gestión basado en una agenda cuyos contenidos son *actos* de

comunicación basados en la teoría de los *Speech Acts* [Cohen and Perrault, 1986].

1.1.4 Lógica Borrosa

La lógica borrosa o difusa (dependiendo de la traducción del término inglés *fuzzy* [Zadeh, 1973]) ha demostrado ser, además de un entorno matemáticamente robusto, una herramienta muy fiable para el desarrollo de controladores eficientes en aplicaciones diversas, desde electrodomésticos a automóviles. Por ello se ha utilizado para el diseño de algunos de los controladores que generan comportamientos básicos empleados en los diferentes experimentos que se han realizado y que se describen en capítulos posteriores.

La propuesta de Zadeh establece básicamente un sistema que permite operar con términos vagos, es decir, con valores de verdad que no son simplemente verdadero y falso, sino un abanico continuo de valores desde el falso al verdadero. Esta variación permite asignar valores de verdad continuos a expresiones del tipo "El sensor_1 **es** cerca", o lo que es lo mismo, valores de posibilidad o probabilidad. Esta teoría permite también aplicar operadores lógicos (conjunción, disyunción, etc.) entre este tipo de expresiones, con lo que se consiguen expresiones como "El sensor_1 es Cerca **Y** el sensor_2 es Lejos". Admite además la definición de calificadores (mucho, poco, etc.) para generar expresiones lógicas del tipo "El sensor_1 es Cerca Y el sensor_2 es **Muy** Lejos". Por último, se pueden realizar implicaciones lógicas mediante una versión del *Modus Ponens*, que asigna valores continuos de verdad a expresiones como "El sensor_1 es Cerca Y el sensor_2 es Muy Lejos **Entonces** motor_1 es Atrás".

La lógica borrosa se ha utilizado en este trabajo para la construcción de algunas de las acciones básicas del robot. Para ello se han implementado diferentes controladores borrosos, cada uno de los cuales generará valores difusos a aplicar en los actuadores del robot en función de la evaluación borrosa de un conjunto de reglas. El Anexo A presenta una descripción más formal de sus componentes y de los procesos de evaluación de las reglas empleados en los controladores. También se han utilizado reglas borrosas para construir las heurísticas que adapten el modo en que se combinan las habilidades básicas a las circunstancias del entorno.

1.1.5 Aprendizaje Automático

Una vez que se dispone de un mecanismo para diseñar comportamientos básicos (p.e. la lógica borrosa) y de un modelo para combinarlos (p.e. el propio modelo ABC^2) hay que plantearse la forma en que se generan dichos comportamientos. En un entor-

no industrial esa generación sería indudablemente heurística, es decir, algún experto implementa los controladores 'ad-hoc' para la aplicación concreta. Sin embargo, en el entorno de la IA siempre existe la posibilidad de explorar mecanismos automáticos de generación o de mejora de dichos controladores.

El área de la IA que se dedica a investigar este campo se denomina Aprendizaje Automático. De nuevo, las dos corrientes dominantes de la IA tienen influencia en las técnicas que se pueden emplear en este campo. Por ejemplo, se pueden utilizar técnicas subsimbólicas como los algoritmos genéticos [Goldberg, 1989] para generar comportamientos básicos de un robot; o bien técnicas simbólicas [Quinlan, 1993] que permiten mejorar la calidad de los planes de alto nivel; o incluso las operaciones conjuntas entre diversos agentes heterogéneos[Giráldez and Borrajo, 1997].

En los distintos experimentos realizados para probar ABC^2 se ha utilizado mayoritariamente el diseño heurístico de las habilidades del robot. También se han realizado experimentos de generación y optimización de los mismos mediante algoritmos genéticos [Matellán *et al.*, 1995a]. De esta forma se ha querido verificar que el modelo admite perfectamente cualquiera de las dos opciones.

Otra posible aplicación de las técnicas de aprendizaje automático utilizando ABC^2 es el aprendizaje de técnicas de cooperación entre distintos agentes. Este tipo de aproximaciones las han aplicado otros autores en uno de los dominios en los que se ha probado el modelo con considerable éxito [Stone and Veloso, 1997a].

1.1.6 Sistemas de Tiempo Real

En los entornos dinámicos en los que trabajan habitualmente los robots autónomos, sus sistemas de control no solo deben tomar las decisiones correctas, sino que además deben tomarlas en el momento adecuado. La investigación en los *sistemas de tiempo real* se dirige precisamente hacia ese objetivo: el desarrollo de métodos que garanticen que la velocidad de reacción del sistema corresponde a la velocidad de cambio en el entorno. En ese sentido, el estudio de los sistemas de tiempo real no se dedica a la realización de sistemas "rápidos", sino a la realización de sistemas "suficientemente rápidos de forma predecible" [Musliner *et al.*, 1993].

El significado de la expresión "tiempo real" en este entorno es radicalmente diferente del uso dado en otros entornos. Por ejemplo, en el acceso a bases de datos, un sistema que es capaz de responder habitualmente en un escala humana de tiempos rápida (p.e. en unos pocos segundos), se considera un sistema de tiempo real. Si en una situación

extraña las condiciones varían (el número de usuarios tiene un brusco aumento), el sistema probablemente se degradará (respondiendo por ejemplo en el orden de las decenas de segundos) de forma que se consideraría no aceptable. Ese es precisamente el comportamiento opuesto al de un sistema de tiempo real, donde se trata de *garantizar* que las diferentes tareas cumplen con sus plazos máximos.

Para conseguir estos objetivos en el campo de los sistemas de tiempo real se han desarrollado un conjunto de técnicas y herramientas para caracterizar las tareas y generar planificadores adecuados. Estos sistemas se han desarrollado en general para controlar sistemas de complejidad limitada y operando en entornos muy estructurados. Por ejemplo, cadenas de montaje, máquinas herramienta, aparatos de electrónica de consumo, etc. Ello ha hecho que sea difícil su expansión a tareas complejas o difícilmente modelizables, como son los entornos de inteligencia artificial.

Por ello, las restricciones de tiempo real no se han tenido en cuenta de forma estricta (lo que se conoce como "tiempo real duro"), pero sí que se ha buscado garantizar que el tiempo de ejecución de los comportamientos básicos usados en ABC^2 puedan ser estimados, siendo éste el primer paso hacia un sistema realmente de tiempo real.

1.1.7 Psicología y Sociología

El campo de la IA se ha relacionado con la psicología desde sus inicios. Esta relación se ha establecido en ambas direcciones; la IA ha tomado conceptos e ideas de la psicología y ésta a su vez ha utilizado a la IA como un campo fundamental de experimentación y prueba. A su vez, la conjunción entre IA y robótica ha aumentado aún más el interés desde el punto de vista de la psicología, pues permite experimentar muchas de sus teorías.

Los planteamientos de las distintas corrientes de la IA han buscado apoyos en las corrientes psicológicas. Por ejemplo, los planteamientos reactivos se han visto apoyados por el modelo de estímulo-respuesta dominante en la psicología de principios de siglo, como la Teoría de los Reflejos Condicionados [Pavlov, 1927] que excluyen la representación interna del mundo y los procesos cognitivos [1], definiendo el comportamiento como una secuencia de reacciones inmediatas a la información sensorial. Por su parte, la IA simbólica ha buscado el apoyo de los cognitivistas que han criticado severamente

[1] Existe cierta polémica en el área de la IA sobre la traducción correcta del término anglosajón *cognitive*. Según el diccionario de la Real Academia, existe el término *Cognitivo-a* (del latín *cognitio -onis*) "relativo al conocimiento'; y *cognoscitivo* (del latín *cognoscere*)"capaz de conocer". En esta memoria se ha optado por el primer término.

dichas ideas [Neisser, 1967, Newell and Simon, 1956]:

> *[...] behavior in general, and human behavior in particular, is not a chain of conditioned reflexes. [...] the effect an event will have upon behavior depends on how the event is represented in the organism's picture of itself and its universe [...][Miller* et al., *1960]*

Otros conceptos tomados de la psicología y cuya consideración ha influido en el diseño de ABC^2 son, por ejemplo, el control *homeostático* [Arkin, 1993] del robot. Es decir, la gestión de la sensibilidad interna del robot, que le informa, por ejemplo, del estado de sus baterías o de posibles fallos en subsistemas. Otros campos de discusión son la importancia de la memoria [García, 1997], de los lenguajes o la sensomotricidad.

Del mismo modo, al diseñar los aspectos relativos a la cooperación entre robots, se han tenido en cuenta las aportaciones de la sociología. Así, se han considerado diferentes modelos para representar las relaciones entre agentes autónomos, eligiendo como modelo más apropiado el de los actos de comunicación [Cohen and Perrault, 1986]. Dicho modelo es una simplificación de los modelos tradicionales de la sociología, en los que se representan las intenciones de los agentes, sus roles, comportamientos sociales (egoístas, colaboradores, etc.), clasificación de las tareas, rendimiento de los grupos, etc.

1.1.8 Biología y Etología

De igual forma que el estudio del comportamiento individual y colectivo del ser humano es una de las bases fundamentales de la IA, el estudio de los sistemas biológicos ha aportado algunos de sus paradigmas a la IA, en especial dentro a la IA subsimbólica. Por ejemplo, la programación genética [Goldberg, 1989], que se ha utilizado en esta tesis doctoral para el aprendizaje de comportamientos básicos, es uno de los muchos paradigmas tomados por la IA de la biología. Otros son las redes neuronales artificiales, los comportamientos emergentes, el cálculo basado en DNA, etc.

En el campo específico de los robots autónomos y desde le punto de vista histórico, la idea de robots que se autogobiernan ha estado ligada a la simulación de formas de vida animal [McFarland and Bösser, 1993]. Así, los primeros ejemplos, basados en las teorías de la cibernética [Wiener, 1948] empleaban técnicas de retro-alimentación negativa. El comportamiento de tales sistemas se basa en la diferencia entre el estado presente y el estado deseado o "meta". Ejemplo temprano de este tipo de sistemas, basados en la emulación del comportamiento animal, son las tortugas de Grey

Walter,[Walter, 1953] alguna de las cuales ha sido recientemente restaurada [2] y que constituyen uno de los primeros ejemplos de robot autónomo tal y como se entiende hoy en día.

La etología también ha hecho sus aportaciones al campo de la IA. Por ejemplo, numerosos investigadores han estudiado e intentado imitar el comportamiento colectivo de ciertos insectos como las hormigas o las avispas [Theraulaz *et al.*, 1990, Steels, 1990]. También se han propuesto diversos sistemas distribuidos en los cuales los comportamientos colectivos emergen como resultado de interacciones locales. Por ejemplo, formaciones de robots que se mueven uniformemente emulando las formaciones de pájaros, por ejemplo las típicas bandadas de vencejos. En ellos, cada individuo toma sus decisiones basándose únicamente en las acciones de sus vecinos más próximos [Arkin, 1992].

1.2 Motivaciones y objetivos

La conjunción entre la IA y robótica da lugar a uno de los campos que despierta más interés en los autores de ciencia ficción [Asimov, 1950, Lucas, 1977] desde hace años. Sin embargo, y a pesar de su popularidad, los avances científico-técnicos en este campo han sido más lentos de lo "vaticinado" por la ciencia ficción. La ilusión por trabajar en un campo tan atractivo, el interés por comprender los problemas que hacen que el desarrollo de la robótica cognitiva sea tan lento y la satisfacción de aportar un grano de arena a su solución, son las motivaciones personales principales de este trabajo. Esta sección está dedicada a presentar estas motivaciones y a exponer los objetivos concretos con más detalle.

El interés científico principal del autor es el control de robots autónomos. Entendiendo por autónomos aquellos robots que toman sus decisiones sin la intervención de un controlador humano. Es decir, los robots que no son teledirigidos. Existe otra definición de "autonomía" que hace referencia a que el robot no dependa de ningún elemento externo a sí mismo. Esta última acepción corresponde a un punto de vista ingenieril, que implica que un robot autónomo acarree sus propias fuentes de energía (baterías en general), toda su circuitería de control (usualmente micro-procesadores y memorias), todos sus sensores, etc. El interés de esta tesis se centra en la primera acepción, aunque el modelo, los robots utilizados para probarlo y los programas

[2] http://www.uwe.ac.uk/facults/eng/ias/walterbot.html

desarrollados se adaptan sin problemas a la segunda.

El objetivo de esta tesis es el desarrollo de un modelo para el control de robots autónomos que permita la ejecución de tareas de alto nivel empleando para ello controladores de bajo nivel, es decir, que integre las dos corrientes de la IA. Pretende también generar un modelo que permita integrar las acciones decididas por el agente o pedidas por otros. Además, estos objetivos se deben cubrir con un modelo que sea:

- *Genérico*, entendido como que el modelo debe ser utilizable tanto por agentes software como por robots.
- *Independiente del hardware*, utilizable en diferentes tipos de robots, proporcionando una abstracción sobre sus componentes.
- *Robusto*, que se degrade suavemente ante modificaciones graduales de su entorno, es decir, que no dependa del conocimiento a priori de su entorno.
- *Tolerante a fallos*, debiendo compensar fallos parciales del sistema y tratar los eventos no previstos.
- *Fiable*, que el sistema funcione correctamente cada vez que se use.
- *Flexible*, que sea capaz de adaptarse (modificar las acciones tanto de un robot individual como de un grupo de robots) a las condiciones de la tarea a realizar, el número de robots involucrados, etc.
- *Coherente*, que el grupo de agentes se comporte como un todo.
- *Extensible*, es decir, que permita integrar habilidades de los robots obtenidas mediante diversos paradigmas: controladores borrosos, comportamientos aprendidos, etc.
- *Multi-agente*, que permita la colaboración de robots heterogéneos y autónomos en la realización de tareas que puedan aprovecharse de la posibilidad de cooperar.

Para comprobar que estos objetivos se han cumplido se planteó como requisito que el modelo desarrollado pudiese funcionar en dominios diversos con el menor número de modificaciones posibles. En particular se ha probado en dominios simulados, utilizando simuladores realizados tanto por el autor y sus compañeros (caso del simulador SimDAI [Sommaruga *et al.*, 1996]), como simuladores desarrollados por otros

investigadores, (como es por ejemplo el simulador de fútbol entre robots de la RoboCup [Itsuki, 1995]); y con robots reales, habiéndose utilizado el mini-robot Khepera [Mondada *et al.*, 1993].

Además se plantea también como objetivo que los dominios de experimentación representen diferentes tipos de tareas. Para ello se presentarán los experimentos realizados sobre dos entornos distintos. El primero representa un entorno típico de cooperación, el traslado coordinado de un objeto por dos robots autónomos. El segundo es un ejemplo de trabajo en un entorno hostil: la competición entre grupos de robots, realizada sobre el simulador de la RoboCup.

1.3 Estructura de la memoria

La presente memoria describe el trabajo realizado por el autor en el diseño, implementación y pruebas del modelo de control de robots ABC^2, haciendo especial hincapié en las aportaciones realizadas por el autor al diseño de este tipo de sistemas. La estructura de esta memoria se basa en tres partes bien diferenciadas.

En la primera, formada por los dos primeros capítulos, se ofrece una visión general del contexto de este trabajo y del estado del arte en el área específica del control de robots. Así, a continuación de la presente introducción, el capítulo 2 se centra en analizar las distintas técnicas de la IA que se han empleado en esta tesis. En la sección 2.1 se analiza la evolución y estado actual de las dos escuelas básicas de la IA, la simbólica (Sección 2.1.1) y la subsimbólica (Sección 2.1.2). Además se hace especial hincapié en los sistemas oportunistas (Sección 2.1.3), donde se enmarca el modelo construido, así como en la influencia de las técnicas multiagente.

La segunda parte se centra en la definición formal del modelo y su implementación. El Capítulo 3 describe el modelo realizado, incluyendo sus componentes (Sección 3.3) y el algoritmo de planificación (Sección 3.4). El Capítulo 4 se dedica a ilustrar la implementación del sistema y los mecanismos previstos para su configuración y uso.

En la tercera parte se describen los experimentos realizados para probar el modelo. En particular, el Capítulo 5 describe los experimentos realizados con dos robots, primero simulados utilizando el simulador SimDAI y luego con dos robots reales. Además, en este capítulo se hace especial hincapié en el diseño de las habilidades básicas de los robots. Se presenta el diseño heurístico, en forma de conjunto de reglas borrosas, y el diseño basado en aprendizaje en forma de algoritmo genético para diseñar un conjunto

de reglas borrosas.

El Capítulo 6 presenta los resultados del uso de ABC^2 en el campeonato mundial de fútbol entre robots, denominado RoboCup, cuya primera edición tuvo lugar en Nagoya (Japón) en Agosto de 1997. Para ello se enumeran en primer lugar las reglas y restricciones de este dominio (Sección 6.1), a continuación los problemas que se deben afrontar (Sección 6.1.2). Se describen los comportamientos empleados (Sección 6.2.1) y se explican los resultados obtenidos (Sección 6.3).

Por último, en el Capítulo 7, se presentan las aportaciones realizadas y las conclusiones extraídas en el transcurso de esta tesis doctoral (Sección 7.1), así como las posibles extensiones que han surgido en su desarrollo (Sección 7.2).

Capítulo 2

Estado del Arte

Creo que releer es más importante que leer,
salvo que para releer se necesita haber leído.
Jorge Luis Borges

El estado del arte en un determinado campo no puede verse como una imagen fija, sino como un proceso en continua evolución. En este sentido, hay que resaltar que desde el final de los años sesenta se ha producido una revolución en el campo del control de robots. Desde una primera generación de robots sin capacidades sensoriales, pasando por la segunda de robots de limitada capacidad computacional y control basado en realimentación, se ha llegado a la tercera generación, formada por robots inteligentes, esto es, dotados de distintas capacidades de decisión en función de sus sensores. Esta posibilidad de tomar decisiones en función de la información percibida del entorno y de la almacenada internamente es la que ha generado el nombre de *robótica cognitiva* para denominar la generación actual de robots.

El presente trabajo se enmarca en el campo de la robótica cognitiva, siendo su objetivo el diseño de un modelo que permita controlar un robot atendiendo a las restricciones básicas de ejecución en un entorno desconocido y de difícil modelización en el que pueden actuar diversos agentes. En este capítulo se traza un breve resumen de las principales técnicas empleadas hasta la fecha, se aporta una visión general de su situación actual y se indica la influencia de los mismos en la definición del modelo ABC^2. Así, en la primera sección se analizan las principales estrategias para el control de robots. A continuación se analizan los problemas relacionados con la existencia de varios agentes en el mismo entorno, presentando las aproximaciones a su solución. Por último, la Sección 2.3 realiza una comparación entre ellas.

2.1 Modelos de Control

Los robots *inteligentes* [Valavanis and Saridis, 1992], se caracterizan en general por ser sistemas complejos controlados por una o más computadoras, que están equipados con distintos sensores y que poseen capacidad de decisión y de resolución de problemas en sus dominios de operación. La modelización de los sistemas de decisión se realiza mediante metodologías heurísticas y analíticas pertenecientes en general a los campos de la teoría de sistemas y a la inteligencia artificial. El enfoque tomado en esta tesis ha sido el segundo, aunque se reconoce cierta influencia de los sistemas de control más tradicionales [Serradilla and Maravall, 1996] basados en formulaciones matemáticas rigurosas, así como de la integración de ambas aproximaciones [Serradilla and Maravall, 1998].

La evolución de estos campos y el empleo de robots en distintas tareas ha ido generando diferentes técnicas de control de robots. El objetivo de esta sección es mostrar cuál ha sido esa evolución y cuál es el estado actual de las corrientes dominantes. Para ello, en esta sección se analiza en primer lugar la planificación simbólica, se continúa con la reactiva y la oportunista y se termina realizando un pequeño resumen comparativo de las tres.

2.1.1 Planificación Simbólica

El desarrollo de la ciencia cognitiva por una parte y de la tecnología informática por otro, hizo que a partir de la década de los sesenta la aproximación cibernética [Wiener, 1948] se abandonase en favor de técnicas cognitivas. Estas técnicas pretenden emular actividades mentales tales como la memoria, las intenciones o los planes. La idea es interponer una capa de software que implemente este tipo de operaciones mentales entre los estímulos sensoriales y las respuestas motoras del robot.

De todas las aportaciones de las ciencias cognitivas, la que ha tenido mayor influencia en el campo de la robótica inteligente ha sido sin duda la resolución simbólica de problemas. Esta aproximación tuvo su origen en los trabajos iniciales de Newell y Simon [Newell and Simon, 1972] sobre resolución de problemas. Su fundamento es la búsqueda de una secuencia de acciones (de entre un conjunto de acciones posibles) tal que sea posible alcanzar un fin predeterminado.

La aplicación de esta técnica al campo de la robótica se plasma en el uso de la planificación simbólica, en el cual el robot debe encontrar una secuencia de acciones que le lleven de un **estado inicial** a un estado en el que se cumplen un conjunto de

metas. Ambos estados se suelen describir de forma simbólica, generalmente utilizando una gramática simple.

Uno de los primeros sistemas de resolución simbólica de problemas fue el *General Problem Solver* (GPS) de Newell y Simon [Newell and Simon, 1963]. La estrategia en la que se basaba se denominó el "Análisis de medios-fines", que consiste en realizar una búsqueda, partiendo del conjunto de metas, hasta el estado inicial.

Una de las aplicaciones más tempranas y conocidas de estas teorías al campo de la robótica fue el robot *Shakey*, [Nilsson, 1969] desarrollado en el *Stanford Research Institute* (SRI) a mediados de los años sesenta. El objetivo era conseguir un robot que se pudiese mover de forma autónoma en un entorno formado por varias habitaciones, empujando objetos predeterminados, y evitando los obstáculos. *Shakey* estaba controlado por un programa denominado STRIPS (*STanford Research Institute Problem Solver*) basado en el análisis de medios-fines [Fikes and Nilsson, 1971].

El mecanismo básico de funcionamiento de STRIPS es el siguiente: se examinan la metas deseadas y se determina el conjunto de acciones (denominados **operadores**) de entre las que era capaz de realizar el robot, que son capaces de hacer que el robot alcance algunas de las metas. Es decir, se genera un conjunto de operadores cuyos efectos (**post-condiciones**) hacen que el robot alcance la meta. Si alguno de ellos es aplicable directamente, la solución será ejecutar esa acción. Sería un **plan** de un solo paso. Ahora bien, en general la solución no es tan simple porque los operadores seleccionados no son aplicables directamente. En ese caso se consideran sus **pre-condiciones**, es decir, qué configuración del mundo haría aplicable cada operador. Para ello, se generan nuevas metas a conseguir (una por cada conjunto de precondiciones del operador considerado). De la misma forma, estas metas se podrían conseguir mediante diversos operadores. Así se genera un árbol donde la raíz es el conjunto de metas iniciales, los nodos son metas intermedias y las transiciones son operadores.

El proceso de generación del plan consiste en buscar un camino descendente desde la raíz a un nodo-hoja cuya definición sea válida en el estado inicial. El plan será la secuencia de operadores que se tienen que ejecutar para pasar del estado inicial a uno en el que sean ciertas las metas. O lo que es lo mismo, las transiciones ascendentes en el árbol desde el nodo hoja hasta la raíz.

Aparte de la complejidad de la búsqueda en este árbol, que crece exponencialmente en función del número de operadores y de factores dependientes de la definición del entorno, hay que considerar la posibilidad de que la ejecución del plan no sea exactamente

la prevista en ese árbol. En el caso de Shakeys la ejecución del plan se encomendaba a un programa denominado PLANEX que se encargaba de comparar los movimientos "planeados" de Shakey con los ejecutados realmente. Si la diferencia entre los dos sobrepasaba un cierto umbral, PLANEX forzaba la actualización del plan, es decir, una replanificación.

Para poder realizar el cálculo de esa diferencia, PLANEX mantenía un **modelo del mundo** en el que se encontraba el robot. Este modelo del mundo se actualizaba en función de la información sensorial proporcionada por el robot. Mantener la coherencia entre el mundo real y su representación hizo que Shakeys fuese extremadamente dependiente del entorno: para que el entorno fuese sencillo de representar y de mantener este debía ser estático y fácil de reconocer, lo que se denomina un **entorno estructurado**. Estas suposiciones se podrían resumir [Fikes and Nilsson, 1971] en:

- El estado del mundo puede ser definido de manera formal y observado sin errores.
- El robot es el único agente que puede producir cambios en el entorno.
- Las acciones del robot tienen sólo los efectos especificados en la definición formal de las mismas.

Uno de los primeros intentos de utilizar las aproximaciones basadas en la búsqueda de planes sobre un modelo del mundo a entornos menos estructurados es el sistema NOAH (Nets Of Action Hierarchies) [Sacerdoti, 1977]. NOAH se diseñó para enfrentarse con los cambios en las posiciones de los objetos en el entorno de Shakey y utiliza los principios de la **planificación no-lineal**. La idea básica es crear una descripción parcial del plan en vez de crear un plan completo, generando el plan a medida que se va necesitando. Así, NOAH genera una jerarquía de acciones parcialmente ordenadas en forma de red. Esta red se pasa al sistema de ejecución y monitorización, donde NOAH dispone de distintos módulos, denominados CRITICS, que se encargan de examinar el plan según se va generando, así como de analizar la información sensorial para comprobar que no se produce una gran disparidad entre la situación prevista en el modelo del mundo y la percibida por los sensores.

Uno de los mayores problemas de los planificadores comentados es que no son capaces de distinguir entre actividades que son críticas para el éxito de un plan y aquellas que son meros detalles. Por ejemplo, planificando un viaje a los Estados Unidos, resulta una pérdida de tiempo considerar el número de calcetines que se van a llevar antes

de pensar en el itinerario o en la compra de los billetes. El método de la *planificación jerárquica* trata de construir en primer lugar un plan abstracto dejando los detalles sin especificar, para luego refinar estos detalles en subplanes. Este tipo de sistemas se han intentado construir siguiendo distintas aproximaciones, como por ejemplo ABSTRIPS [Sacerdoti, 1973], generalmente para dominios muy abstractos. Otro ejemplo, es uno los primeros trabajos del autor en el campo de la robótica, consistente en el control de las acciones de un manipulador (brazo con seis grados de libertad) [Mayo, 1996]. En él se utilizaba el sistema PRODIGY [Veloso *et al.*, 1995] para generar un plan a alto nivel, sin tener en cuenta el entorno circundante. A continuación, algunos de los operadores del plan generado se replanificaban considerando la descripción detallada del entorno.

Los sistemas basados en búsquedas heurísticas sobre un espacio de estados creados a partir de una representación del entorno han ido evolucionando con el paso del tiempo hasta generar arquitecturas completas multi-propósito, como PRODIGY o SOAR [Lehman *et al.*, 1996]. Este tipo de arquitecturas están generalmente basadas en un planificador y una serie de módulos que lo extienden para su uso en dominios particulares. Sin embargo, no son muchos los usos de arquitecturas cognitivas genéricas en robots reales. Ello es debido, en general, a la severa crítica a la que se han visto sometidos estos sistemas desde finales de los años 80 [Brooks, 1991b, Agre and Chapman, 1990].

El principal argumento de dicha crítica es el elevado coste computacional que exigen este tipo de sistemas, para poder tratar los problemas de incertidumbre y dinamismo del mundo real. Sin embargo, es probable que los enormes avances en el hardware de los computadores permitan la aparición de robots capaces de utilizar satisfactoriamente este tipo de aproximaciones. El trabajo teórico en ese campo se basa generalmente en incluir en el planificador una representación probabilística explícita de los posibles eventos (lo que exige de nuevo poder modelizarlos). Por ejemplo, se pueden representar los eventos de forma similar a los operadores de un planificador tradicional, con sus efectos sobre el mundo, incluyendo una probabilidad de ocurrencia [Blythe, 1996].

Un ejemplo del tipo de tareas que pueden llegar a realizar los robots controlados actualmente mediante técnicas basadas en un modelo interno del mundo, podría ser el robot RHINO [Thrun, 1996, Thrun *et al.*, 1997]. Este robot tenía como misión realizar visitas guiadas en una exposición particular de un museo alemán a las personas que se lo solicitasen, para lo cual disponía de un CD que debía reproducirse en los lugares adecuados. Para llegar hasta ellos debía moverse en un entorno real, lleno, por ejemplo, de impredecibles niños; y hacerlo suficientemente deprisa como para resultar útil.

El sistema construido partía de una representación topográfica del entorno, generada a priori, en la que se almacenaban los límites de la sala y los objetos fijos. Dicha representación se realizaba en función de una rejilla (*grid*). Un planificador generaba las rutas usando este mapa e iba pasando las metas a un módulo "reactivo" que se encargaba de ejecutar las acciones básicas. Es decir, aunque era un sistema fundamentalmente basado en la corriente cognitiva, utilizaba conceptos de los sistemas reactivos que se analizan en la siguiente sección.

Por último, indicar que el campo de la planficación simbólica sigue evolucionando, apareciendo nuevos planificadores como SNLP [McAllester and Rosenblitt, 1991] o UCPOP [Penberthy and Weld, 1992] que aportan diferentes algoritmos que permiten optimizar el proceso de búsqueda. Así mismo, existen otras adaptaciones de los principios de la planificación simbólica que permiten, por ejemplo, razonar sobre la duración de las acciones (planificación temporal), añadir cierto tratamiento de la incertidumbre, etc. Muchos de estos sistemas parecen prometedores pero todavía no han sido empleados con profusión en el campo de la robótica, por lo que no se han detallado en esta descripción.

2.1.2 Sistemas Reactivos

Las limitaciones de la aproximación simbólica devolvió protagonismo al final de la década de los ochenta a las aproximaciones subsimbólicas. Así, resurgió un movimiento que se puede definir como reactivo [Brooks, 1991a] que sostiene que es posible diseñar sistemas inteligentes sin necesidad de mantener una representación interna del mundo externo. Según esta corriente, basta con diseñar procesos simples, cada uno con su propio bucle de percepción-reacción, y combinar su ejecución adecuadamente. Estos procesos pueden estar basados en distintas técnicas de la IA como las redes neuronales [Weitzenfeld *et al.*, 1997], las máquinas de estado finito en la *subsumption architecture* [Brooks, 1986] o los controladores borrosos [Matellán *et al.*, 1995b].

La idea básica de esta aproximación sostiene que el comportamiento en un momento determinado debe ser controlado por la situación existente en ese instante. Los mecanismos elaborados de construcción de modelos internos del mundo y las inferencias de situaciones futuras sobre él deben evitarse. Esta aproximación se asocia generalmente al grupo de Rodney Brooks en el Massachusetts Institute of Technology, aunque aproximaciones similares pueden encontrarse en trabajos contemporáneos como el autómata de Kaelbling [Kaelbling, 1987] o en el jugador reactivo de vídeo-juegos (SONJA) de

Agre y Chapman [Agre and Chapman, 1990].

La construcción del sistema, dadas sus metas, acciones potenciales y estados relevantes posibles del entorno, consiste en definir una serie de reglas de percepción-acción específicas para esas metas, y almacenarlas de forma computacionalmente eficiente (en forma de máquinas de estado finito en el caso del robot andante de Brooks [Brooks, 1986]). Para cada iteración del ciclo percepción-acción, el agente percibe el estado del entorno, dispara la regla asociada y realiza su acción.

Formalización 2.1 Máquina de estado finito para evitar un obstáculo

```
1   (defmodule runaway
2     :inputs (force)
3     :outputs (command)
4     :states
5       ( (nil
6           (event-dispatch force decide))
7         (decide
8           (conditional-dispatch(significant-force-p force)
9                                runaway
10                               nil))
11        (runaway
12          (output command (follow-force force))
13          nil)))
```

Uno de los ejemplos más conocido de arquitecturas de este tipo es el robot Herbert desarrollado por Jonathan H. Connell [Connell, 1990]. Su misión era deambular por un entorno típico de oficinas, detectar latas de coca-cola, verificar si estaban vacías y si lo estaban llevarlas a un lugar prefijado. El sistema está constituido por una serie de módulos, basados en la *subsumption architecture*, para tratar las distintas situaciones. Por ejemplo, la definición del módulo que hace que el robot evite un obstáculo se muestra en la Formalización 2.1, donde se describe una máquina de estado finito con tres estados: *nil*, *decide* y *runaway*. Además, se representa el sistema mediante una entrada, *force*, que corresponde a un valor que significa una fuerza de repulsión, por ejemplo la distancia a un obstáculo, y una salida, *command*, que representa un comando. La máquina permanecerá en el estado *nil* hasta que se detecte un evento en la entrada *force*, en cuyo caso se pasará el control al estado *decide* (línea 7). En ese estado se comprueba si el valor de *force* es significativo o no (línea 8); si lo es, se pasa el control al estado *runaway* y, si no, se devuelve a *nil*. Por último, el estado

runaway únicamente se encarga de ejecutar el comando correspondiente, por ejemplo, `follow-force`.

La implementación de este tipo de sistemas se ha hecho muy popular desde finales de los años 80. Así, distintos trabajos han mostrado que este tipo de arquitecturas puede construirse utilizando técnicas variadas. Por ejemplo, Ruspini y Safiotti [Saffiotti and Ruspini, 1993] han utilizado comportamientos construidos mediante lógica borrosa, otros han empleado redes neuronales, etc.

Con un sistema de este tipo el robot puede responder con gran rapidez a la mayoría de las situaciones que se producen durante la ejecución. Sin embargo, tiene el problema de que hay que identificar, almacenar y anticipar todas las posibles contingencias. En otro caso se producirán comportamientos no deseados ante condiciones no anticipadas. Además, el hecho de determinar el comportamiento del agente atendiendo únicamente a razonamientos locales compromete el rendimiento global del sistema, la eficiencia y la coherencia con la meta global. Por ejemplo, un agente que se encuentra en una situación crítica comienza a ejecutar acciones localmente adecuadas, pero globalmente descoordinadas con la meta global. Es lo que se denomina habitualmente un mínimo local.

Irónicamente, el debate entre los partidarios de los sistemas reactivos y los planificados reproduce un debate similar entre los psicólogos, sólo que en orden inverso. Durante la primera mitad del siglo la psicología estuvo dominada por el modelo de "Estimulo-Respuesta (E-R)" [Pavlov, 1927]. Como el modelo reactivo, excluyen la representación interna. A cambio, modelan el comportamiento como una secuencia de reacciones inmediatas a los estímulos externos. Sin embargo, y reconociendo que tales reacciones ocurren, la mayoría de los psicólogos acabó por concluir que el modelo E-R por sí solo no podía explicar todos los experimentos, ni el comportamiento humano, ni siquiera todos los comportamientos de los animales superiores.

La disputa entre los partidarios de ambas opciones ha acabado, como era de esperar, en la generación de modelos híbridos. Esta corriente ha partido especialmente del dominio de la robótica, donde la corriente reactiva tiene gran fuerza por la facilidad para crear aplicaciones concretas. Una de estas opciones híbridas es la planificación oportunista que se describe en la próxima sección y en la que se basa ABC^2.

2.1.3 Planificación Oportunista

Tanto en los modelos basados en la planificación, como en los basados en la reacción, los agentes deciden anticipadamente las acciones a ejecutar para conseguir las metas definidas. Los modelos basados en la planificación explotan la predecibilidad del entorno para eliminar las tareas de monitorización y para restringir las acciones a una sola secuencia de acciones coordinada globalmente. Sin predecibilidad, el modelo reactivo lo que hace es preparar un elevado número de acciones para todas las posibles contingencias.

Cuanto más especializados se hacen los operadores de un sistema de planificación, o más numerosos los comportamientos de un sistema reactivo, más difícil se hace el problema de en qué orden aplicarlos. Existen diversas formas de tratar este problema. Una de ellas consiste en especificar para cada operador las condiciones bajo las cuales se debería usar. De esta forma, los operadores se aplican ahora de forma no determinista, en función de la situación actual. A este tipo de sistemas se los conoce habitualmente como "planificadores oportunistas" [Hayes-Roth *et al.*, 1979].

Uno de los primeros modelos basado en este tipo de ideas es la arquitectura conocida como de Pizarra (*Blackboard* [Hayes-Roth, 1985]). Barbara Hayes-Roth, la promotora del mismo, establece que el modelo de control oportunista [Hayes-Roth, 1992] está compuesto de tres partes. La primera, "disparada por eventos", preserva las virtudes del modelo reactivo. La segunda está basada en un "planificador estratégico", que mantiene las ideas de la planificación . Por último, un tercer elemento denominado "proceso de control" se encarga de casar las acciones disparadas con las que satisfacen el plan global.

Disparar las acciones en función de los eventos que se producen en tiempo de ejecución conserva una de las cualidades más importantes de los sistemas reactivos: su capacidad para aplicar las acciones en el momento más adecuado, que sólo puede descubrirse durante la interacción del agente con el entorno. Sin embargo, el modelo reactivo ofrece sólo una versión limitada de esta capacidad, puesto que una y sólo una acción está disponible para la ejecución en cada instante de tiempo. Esto es así porque los sistemas reactivos establecen de forma invariable la jerarquía de metas del agente.

El sistema oportunista utiliza condiciones de activación para disparar las reglas, lo que permite identificar distintas acciones posibles para cada instante. Dependiendo de sus metas, un agente puede o no ejecutar una acción. Es decir, un agente puede cambiar sus metas sin cambiar la forma en que dispara sus acciones. Esto permite que

el agente pueda ejecutar acciones que están más allá de sus metas instantáneas. Esta es, en suma, una forma de responder al entorno que ni los sistemas reactivos, ni los planificados proporcionan. En resumen, en un modelo oportunista, *los eventos activan, pero no controlan, las acciones del agente.*

Barbara Hayes-Roth describe diversas aplicaciones prácticas de su sistema, como por ejemplo, la monitorización de unidades de cuidados intensivos, modelado de proteínas, tutoría inteligente, etc. En general se puede decir que el modelo oportunista puede resultar útil en entornos donde sea necesario adaptar velozmente las metas a las demandas y oportunidades que se producen durante la ejecución.

James Firby [Firby, 1994] propone una aproximación distinta a los sistemas oportunistas. El sistema que propone se denomina RAP (*Reactive Action Packages*). Su idea es permitir la ejecución reactiva de planes simbólicos. Para ello se basa en la definición de tareas (*Tasks*) descritas como programas sensibles al entorno que denomina RAPs. Por ejemplo, el RAP que describe cómo coger algo en un entorno simulado de distribución se muestra en la Formalización 2.2. Este sistema está escrito en Common Lisp y se encuentra disponible para diversas arquitecturas, en forma de entorno integrado para la definición y prueba de tareas.

Formalización 2.2 Definición de un RAP para coger un objeto con un brazo

```
1   (define-rap (arm-pickup ?arm ?thing)
2     succeed (ARM-HOLDING ?arm ?thing)
3     (method
4       (context (not (TOOL-NEEDED ?thing ?tool true)))
5       (task-net
6         (t1 (arm-move-to ?arm ?thing) (for t2))
7         (t2 (arm-grasp-thing ?arm ?thing))))
8     (method
9       (context (TOOL-NEEDED ?thing ?tool true))
10      (task-net
11        (t1 (arm-pickup ?arm ?tool) (for t2))
12        (t2 (arm-move-to ?arm ?thing) (for t3))
13        (t3 (arm-grasp-thing ?arm ?thing)))))
```

Este RAP declara dos métodos (líneas 3 y 8) para conseguir la meta. La cláusula de la línea 2 comprueba sobre la memoria del sistema si dicha meta se ha conseguido o no. Para conseguirla, cada uno de los métodos consta de un plan predefinido (que se denomina *task-net*) y que comienzan respectivamente en las líneas 5 y 10. Estos

planes están formados por una secuencia de acciones (nombradas como `t1`, `t2`, etc. en la Formalización 2.1). Es decir, las *task-net* constituyen una forma de organizar las respuestas reactivas [Firby, 1996]. Cada uno de los planes viene condicionado por un *context* que indica cuál o cuáles de los métodos son aplicables en un momento dado.

El algoritmo de un sistema RAP selecciona una tarea para su ejecución. Si es una acción primitiva (sin subtareas) la ejecuta directamente. Si es un RAP, comprueba si su cláusula *succeed* se cumple. Si lo hace, se la considera finalizada y se pasa a ejecutar otra tarea. Si no lo está, se comprueban las cláusulas *context* para ver cuál de los métodos es aplicable. Se selecciona uno y se introducen sus subtareas en una agenda de tareas pendientes de ejecutar. Si después de haber ejecutado todas las subtareas la condición *succeed* sigue sin cumplirse, se selecciona otro de los métodos aplicables y así sucesivamente.

Otra aproximación diferente son los programas T-R (*Teleo-Reactive*) propuestos por Nils Nilsson [Nilsson, 1994]. En su forma más simple un programa TR consiste en una lista ordenada de reglas de producción $(K_1 \rightarrow a_1), (K_2 \rightarrow a_2), \ldots (K_i \rightarrow a_i) \ldots (K_n \rightarrow a_n)$, donde las K_i son condiciones basadas en un modelo del mundo o directamente en percepciones del entorno. Las a_i son acciones en el mundo o modificaciones en el modelo del mundo. Esta lista de acciones se evalúa comenzando por la primera, ejecutando la primera acción cuya condición se cumpla.

Según este modelo los programas TR serían parecidos a los sistemas de producción propuestos hace tiempo. Sin embargo, son muy diferentes. Las acciones a_i descritas en los TR son acciones "durativas", no discretas. Por ejemplo, un robot puede ejecutar la acción "durativa" avanzar (a una velocidad constante) indefinidamente. Así, una acción "durativa" se define como una acción que permanece mientras su condición sea verdadera, lo cual implica que las condiciones se tienen que estar evaluando continuamente.

Este formalismo básico se extiende para permitir condiciones compuestas, donde los TR se representan como árboles, y donde si una acción a_i proporciona o consigue K_j se denotará de forma que $j < i$. Así, se llega a generar un árbol cuya raíz se denomina *nodo meta*. En el árbol, si dos o más nodos tienen el mismo padre, significa que existen dos formas distintas de conseguir la condición de activación del padre. La ejecución del sistema consiste en la evaluación de las condiciones de los nodos del árbol, ejecutando aquel nodo de menor profundidad.

Otro sistema calificable de oportunista, aunque está basado en un planificador no-

lineal tradicional, es O-Plan [Currie and Tate, 1991]. La historia de O-Plan comienza con un punto de vista de ingeniería del software, tratando de construir una arquitectura de control y planificación modular y abierta. Dicha arquitectura debería permitir la experimentación y el reemplazamiento dinámico de alguno de sus componentes así como el uso de información detallada del dominio.

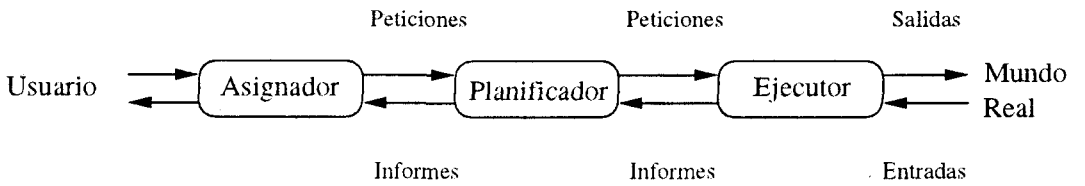

Figura 2.1: Arquitectura de O-Plan

El sistema diseñado, actualmente en su segunda versión, consta como muestra la Figura 2.1, de tres "agentes" o módulos independientes: un *asignador* de tareas, un *planificador* y un *sistema de ejecución*, que se denominan también planificador estratégico, táctico y operacional, respectivamente.

El funcionamiento del sistema es el siguiente: un usuario especifica una tarea mediante una interfaz adecuada que le proporciona el *asignador* de tareas. El *planificador*, que tiene cierta información sobre el sistema de ejecución y sobre el dominio, genera un plan que se pasa al *sistema de ejecución* que se encarga de ejecutarlo. El *sistema de ejecución* ejecuta el plan en el mundo real y monitoriza su ejecución. Si detecta algún fallo (efectos indeseados de acciones, eventos imprevistos, etc.) puede intentar corregirlo o devolverlo al planificador central.

La diferencia con los planificadores tradicionales es la estructura del plan generado. El plan contiene *flaws* (literalmente "defectos, desperfectos, grietas") que permiten modificar el plan generado. Estos *flaws* pueden ser informaciones adicionales, acciones de nivel superior, enlaces a otros planes, etc. Estos elementos son procesados mediante *Knowledge Sources* (KS) similares a las de la arquitectura de pizarra. Se emplea para ello una agenda con los *flaws* pendientes, la cual se maneja con un controlador encargado de asociar los eventos producidos con los *flaws* correspondientes y con las KS capaces de procesarlos. En resumen, se puede decir que O-Plan es un sistema basado en un planificador tradicional que utiliza una agenda para conmutar entre diferentes opciones pre-planificadas de forma oportunista.

Para cerrar el resumen de este sistema, indicar que el esfuerzo principal del proyecto O-Plan fue en el área de la generación de los planes. La continuación del mismo,

denominada O-Plan2 [Tate *et al.*, 1994], se ha centrado más al estudio de las relaciones entre los tres agentes que forman el sistema.

Otro tipo distinto de modelos para el control de robots son los basados en la descripción formal de las duraciones de las tareas. Esta es la aproximación seguida en CIRCA (*Cooperative Intelligent Real-Time Control Architecture*) [Durfee *et al.*, 1997], donde los controladores, denominados TAP (*Test-Action Pairs*) son parecidos a los operadores oportunistas, incorporando duraciones, como muestra la Formalización 2.3.

Formalización 2.3 Ejemplo de un TAP para evitar colisiones

```
1  TAP stop-if-object-ahead
2      :TEST (and *moving* (< (sonar-reading) *safety-distance*))
3      :ACTION (progn (halt) (notify-AIS 'halted))
4      :RESOURCES (sonar base-motors)
5      :MAX-PERIOD .70  :TEST-TIME .15  :ACTION-TIME .05
```

TEST especifica que el TAP sólo se deberá ejecutar si el robot se está moviendo y la distancia delante del robot, medida por el sonar, es menor que `*safety-distance*`. Si dichas condiciones son ciertas, el robot se detiene y se envía un mensaje al AIS (*Artificial Intelligence Subsystem*). El AIS es el encargado de descomponer una meta en planes compuestos de diversas fases y de generar los TAP adecuados para cumplirlas. Para ello utiliza un planificador tradicional.

La particularidad de este tipo de sistemas es que garantizan la consecución de acciones en ciertos plazos de tiempo. Así, la línea 5 de la Formalización 2.3 especifica que el tiempo máximo necesario para evaluar y ejecutar esta acción es de 0.20 segundos (TEST-TIME + ACTION-TIME) y también que el AIS ha determinado que este TAP debe evaluarse, como máximo, cada 0.70 segundos para garantizar que el robot no choca. Es decir, la arquitectura se puede definir como oportunista en cuanto a su (*scheduler*), pero su fundamento son los sistemas de tiempo real.

Existen otros modelos, clasificables como oportunistas, que están a medio camino entre el tiempo real duro de CIRCA y los sistemas de alto nivel como RAP o TR. Un ejemplo podría ser la arquitectura TCA (*Task Control Architecture*) de Reid Simmons [Simmons and Mitchell, 1989]. Esta arquitectura proporciona en primer lugar un acceso ordenado a los recursos del robot de forma que no es necesario construir un *scheduler* para cada aplicación. Además, permite la definición de "comportamientos deliberativos", es decir, planificadores tradicionales, a los que se añaden ciertas restricciones de

tiempo [Simmons, 1992]. Por último, permite utilizar comportamientos "reactivos", implementados como excepciones, cuyas condiciones de ejecución se evalúan cada cierto tiempo. Su aportación principal es permitir que los comportamientos se añadan después de construidos los deliberativos. Es decir, permite una construcción estructurada del sistema [Simmons *et al.*, 1997].

Por último, citar que existen modelos como INTERRAP (Integration of Reactive behavior and Rational Planning) [Muller and Pischel, 1994], que combinan la integración de reactividad y planificación de una forma similar a la oportunista con el tratamiento de los sistemas multi-agentes. Estos sistemas se analizarán en la Sección 2.2.

En definitiva, cada una de las filosofías propuestas para el control de robots tiene sus ventajas y desventajas, y cada una de las implementaciones concretas trata de aprovechar las primeras y mitigar las segundas de la mejor manera posible.

2.1.4 Resumen

Las características de los tres tipos principales de control de robots que se han expuesto pueden resumirse en la Tabla 2.1.

Modelo	Representación	Suposiciones	Metas (tareas)
Simbólico	Mundo y operadores	Mundo estático	Flexibles y explícitas
Oportunista	Habilidades	Acciones preplanificadas	Flexibles e implícitas
Reactivo	Sin modelo	Markov	Prefijadas e implícitas

Tabla 2.1: Resumen de los modelos de planificación.

Bajo las suposiciones de los sistemas simbólicos, el módulo de control recibe una descripción completa del mundo y las potenciales acciones que el robot puede realizar. Entonces, suponiendo que el mundo sólo se modifica según los efectos definidos en sus operadores y que conoce la situación inicial, se le puede especificar una meta como la configuración deseada del mundo. Esta forma de especificar las metas es muy flexible, pues sin variar la definición de los operadores ni del mundo se pueden definir cuantas metas se deseen. El papel del planificador es generar la secuencia de operadores que llevan desde estado inicial a la meta pedida.

Por contra, los sistemas reactivos no emplean ninguna representación del mundo, por tanto, el sistema anterior para especificar las metas a conseguir no es válido. Es más, los sistemas reactivos carecen del concepto de meta. Los objetivos se establecen de forma implícita en la especificación de las acciones posibles del agente y de las

relaciones entre ellas.

La definición de estas acciones consta de dos partes: la condición de activación, y la acción a realizar. Las relaciones entre las acciones establecen su prioridad de ejecución, de forma que la activación de una acción inhibirá la activación de todas aquellas menos prioritarias. Este mecanismo de inhibiciones se establece 'a priori' y permanece inalterado durante toda la ejecución, lo que hace que las metas estén prefijadas de antemano y sean implícitas.

La ejecución de las acciones se realiza mediante su activación, sin tener en cuenta las acciones ejecutadas previamente. Es decir, el sistema carece de memoria, lo que también se suele enunciar diciendo que en general los sistemas reactivos asumen las hipótesis de Markov.

El modelo ABC^2 se puede definir como un sistema oportunista, formado por dos niveles. El primero está compuesto por comportamientos que son capaces de controlar el robot de una forma reactiva y que por tanto carece de representación del mundo. El segundo utiliza un sistema simbólico basado en una agenda para combinar dichos comportamientos. Esta agenda gestiona los comportamientos del primer nivel (denominados habilidades en ABC^2). Por tanto, se puede decir que la representación que utiliza el modelo oportunista, en el caso concreto de ABC^2, está basada en la descripción de las habilidades, que a su vez sirven para definir las metas (como se analiza en el Capítulo 3). O lo que es lo mismo, el modelo oportunista planteado disponde de metas flexibles e implícitas definidas por medio de las habilidades.

En el caso de ABC^2 hay que considerar las particularidades que le aporta el hecho de ser un modelo diseñado para entornos multi-agente, algunas de las cuales se analizan en la siguiente sección.

2.2 Sistemas Multi-Agente

Los primeros sistemas de control de robots se enfrentaban al problema, de por sí complejo, de tomar decisiones referidas a un robot que operaba aisladamente en un determinado entorno. En esas circunstancias la tarea que debía realizar el robot era individual y los medios para conseguir desarrollarla eran sus propias acciones. El abaratamiento del hardware, que hacía posible disponer de más robots, la importancia de conseguir redundancia en las tareas asignadas a los robots, para conseguir mayor fiabilidad, rapidez, etc., y las ventajas que en general aporta la división y especialización del trabajo,

han hecho surgir la necesidad de disponer de sistemas de control que aborden este problema.

Desde el punto de vista de la IA Distribuida [Bond and Gasser, 1988], este tipo de problemas caen dentro del área denominada "Sistemas Multi-Agente". En dicha área se han realizado durante los últimos años multitud de trabajos. Por ejemplo, el desarrollo de modelos teóricos de las acciones y las interferencias entre ellas, algunas de las cuales pueden ser dañinas (como el caso de las que producen un inter-bloqueo entre dos robots) mientras que otras muy similares pueden ser beneficiosas (levantar un objeto por presión en lados opuestos, siguiendo con el ejemplo). Otro ejemplo es el desarrollo de modelos de planificación, como la planificación cooperativa, la planificación en presencia de adversarios, la planificación con contingencias, etc. O también, el estudio de sistemas capaces de razonar sobre las creencias, objetivos, intenciones, etc. de las entidades participantes a partir de sus comportamientos, y así sucesivamente. Las aplicaciones de estos trabajos teóricos han ido desde el reconocimiento del habla, el control del tráfico aéreo, a la gestión de sensores distribuidos y a otras áreas.

De todos estos trabajos, el que más importancia ha tenido en el diseño del modelo propuesto es la arquitectura COOPERA [Sommaruga and Catenazzi, 1996]. En concreto, el modelo utiliza una versión simplificada de los actos de comunicación implementados en COOPERA [Sommaruga, 1993], que a su vez utiliza los fundamentos de los actos de comunicación [Cohen and Perrault, 1986]. Básicamente, la idea se podría resumir en añadir al concepto clásico de operador (formado por sus precondiciones, efectos y cuerpo que genera esos efectos), la facilidad para que las definiciones del mundo de otros agentes puedan tener en cuenta su existencia. Por ejemplo, para hacer suposiciones sobre su ejecución, realizar peticiones, enviar informaciones, etc.

Desde el punto de vista de la robótica, se puede decir que existen dos aproximaciones básicas a la hora de abordar la cooperación de robots móviles autónomos y que reflejan las corrientes básicas de la IA: la cooperación **intencional** y la cooperación **emergente**. La primera supone que los robots tienen un alto grado de inteligencia, es decir, pueden realizar tareas significativas individualmente. Para ello, en general, utilizan un mecanismo de planificación tradicional. La segunda estudia el comportamiento de un grupo de robots generalmente homogéneos, que individualmente tienen pocas capacidades, generadas habitualmente con planteamientos puramente reactivos, pero que cuando se combinan con otros pueden generar comportamientos más complejos (de ahí el nombre de emergente).

La primera suele asociarse con planteamientos basados en la IA simbólica, mientras que la segunda suele utilizar planteamientos de la IA subsimbólica.

Según esta taxonomía, el modelo ABC^2 se clasificaría dentro en la primera categoria. Otro ejemplo dentro de esta categoría es el propuesto por Fabrice R. Noreils [Noreils, 1993]. Se trata de una arquitectura compuesta de tres niveles. El primero, denominado de *planificación*, se encarga de la cooperación y la coordinación de los robots. En el módulo de cooperación se descomponen las tareas y se utiliza un mecanismo similar al *Contract Net* [Smith, 1980] para su asignación. El robot que realiza la descomposición se convierte en el líder del grupo que se forma. De igual forma, el robot que recibe una subtarea se convierte en el líder del grupo encargado (en caso de que no la pueda realizar él solo) de realizarla. En el módulo de coordinación se gestiona la sincronización de los robots en aquellas tareas que lo requieren. El segundo nivel es el de *control*, que organiza los módulos de un robot para obtener un comportamiento consistente. El tercer nivel es el denominado de *funcionamiento* y se encarga de proporcionar las funciones básicas de procesamiento, detección de eventos y reactividad programable. Como prueba de esta arquitectura se presentan dos robots que realizan el traslado conjunto de una caja, así como una aplicación del modelo al transporte en forma de convoy.

En [Caloud *et al.*, 1990] se describe otra implementación de este tipo de arquitectura. En este caso el modelo consta de un planificador de tareas, un asignador de tareas, un planificador del movimiento y un monitor de ejecución. Cada robot fija sus metas, bien en función de su situación particular, o bien, por la petición de otro miembro del equipo. Utiliza redes de Petri para la interpretación de la descomposición del plan y la monitorización de la ejecución.

Otra arquitectura basada en conceptos similares es *ACTRESS* [Asama *et al.*, 1991]. Esta arquitectura se basa en la distribución de las tareas durante el diseño del sistema y en el uso de la cooperación durante la fase de operación del sistema. De esta forma, el operador humano encarga una tarea al sistema y cada robot selecciona la que realizará. Cuando un robot necesita la colaboración de otro robot emite una petición a otro robot [Asama *et al.*, 1992]. Cada robot chequea las peticiones de los otros robots dando prioridad a la colaboración en las tareas pendientes de los demás antes de pedir ayuda para las propias. Como demostración de esta arquitectura han realizado un sistema prototipo consistente en dos robots que empujan una caja por un pasillo con obstáculos [Ishida *et al.*, 1994].

Por su parte en [Cohen *et al.*, 1990] se propone una subdivisión jerárquica de la autoridad para solucionar problemas distribuidos. El ejemplo de prueba elegido es la extinción cooperativa de incendios. El sistema, denominado *Phoenix*, es un planificador adaptativo, que incluye un entorno de simulación genérico. Basándose en esa simulación el controlador principal en esta arquitectura, denominado *Fireboss*, obtiene una visión del entorno, realiza planes globales y envía instrucciones a los robots para activar sus planificadores particulares.

Otro concepto diferente de sistema multi-agente intencional, basado completamente en planteamientos tradicionales de planificación, es el seguido por arquitecturas como STRPLAN [Llavorí, 1996] o la versión multi-agente de PRODIGY [Pérez, 1991]. Estos modelos utilizan un planificador simbólico para generar de forma centralizada un plan global. Este plan se descompone en sub-planes (por ejemplo por regiones en el caso de STRPLAN), confiando la ejecución de los subplanes a los distintos agentes.

En cuanto a la vertiente emergente, se pueden citar como ejemplo los trabajos sobre estrategias para la recolección y transporte de objetos como los propuestos en [Deneubourg *et al.*, 1990], donde los comportamientos cooperativos emergen a partir de comportamientos básicos. O los basados más directamente en los descubrimientos de la etología como los de Guy Theraulaz en [Theraulaz *et al.*, 1990], que describe estrategias de control cooperativas basadas en el estudio de las colonias de avispas, aplicadas también al problema de recolección. Luc Steels en [Steels, 1990] presenta estudios de simulaciones del uso de varios sistemas dinámicos (con movimiento parcialmente aleatorio) para conseguir funcionamiento emergente. Estos estudios se aplican al problema de la recolección de muestras de roca en un planeta lejano. Por su parte, Alexis Drogoul y Jaques Ferber [Drogoul and Ferber, 1992] describen comportamientos simulados de formación de hileras de robots y de recogida de forraje.

Muy conocidos son también los experimentos de Maja Mataric [Mataric, 1992] sobre comportamientos grupales tales como dispersión, agregación, etc. realizados en el MIT con un grupo de 20 robots reales. Otro trabajo experimental interesante se presenta en [Kube and Zhang, 1990], donde se describe cómo un grupo de cinco robots localiza y empuja de forma emergente una caja. De forma similar en [Stilwell and Bay, 1993] se presenta un método para controlar emergentemente un grupo de robots para resolver el problema del transporte colectivo de una carga paletizada.

Por último, comentar que existen otros conceptos de cooperación entre robots, como por ejemplo el propuesto en el proyecto *CEBOT* [Fukuda and Ueyama, 1994], donde

no se persigue que un grupo de robots colabore más o menos en una tarea, sino que se busca que una colección de robots compongan dinámicamente diversas configuraciones (un robot mayor) según la tarea a desarrollar. Es decir, que los propios robots se generen físicamente a partir de elementos más pequeños.

2.3 Discusión

La aproximación de los sistemas simbólicos basada en búsquedas en un espacio de estados, ha sido y es muy popular, concentrando sin duda la mayor parte del esfuerzo investigador en este área. Es fácil de entender y se adapta al modelo de funcionamiento de nuestra mente (al menos al que suponemos). Además, en problemas "sencillos", optimizados, bien diseñados y que constan en general de una tarea única, producen resultados satisfactorios. Sin embargo, los entornos multi-tarea, o los no bien definidos (donde hay muchos factores imprevistos o una elevada incertidumbre en los sistemas de percepción o de ejecución de los operadores) son difícilmente abordables con este tipo de aproximación. Se pueden resumir los problemas principales de los sistemas simbólicos en:

- Necesidad de una capacidad de cálculo muy elevada (lo que implica en general necesidad de tiempo elevado de cálculo) para realizar la búsqueda en el árbol.
- Ausencia de la consideración de los posibles errores en la ejecución de los operadores.
- Existencia de problemas de coherencia entre el modelo interno y el mundo real, lo que obliga a realizar simplificaciones del mundo para poder modelizarlo.
- Gasto innecesario de recursos en mantener el modelo del mundo.
- Costosa reacción, generalmente replanificación de las acciones, ante eventos no previstos en una secuencia de acciones.
- Incremento exponencial del coste de planificación al enfrentar sistemas dinámicos o multi-agente.

Es decir, su principal problema es el tratamiento de la incertidumbre y las contingencias. O realizan ese tratamiento, resultando demasiado costosos (el árbol de búsqueda es aún mayor) para poder usarse en entornos reales; o no la consideran,

siendo también imposibles de usar en entornos reales por falta de fiabilidad. Como consecuencia, los entornos donde se han utilizado este tipo de sistemas han exigido la simplificación (estructuración) del mundo.

Por su parte, los sistemas reactivos presentan también una serie de problemas:

- La ausencia de metas explícitas hace difícil la ejecución de tareas de alto nivel.
- Todos los modelos existentes están orientados a una aplicación concreta, lo que hace difícil y costoso su diseño y sobre todo su adaptación a nuevos entornos o tareas.
- No mantener una representación interna del entorno impide comunicar información sobre él, lo que es un impedimento a la hora de compartir información entre varios robots.
- La ejecución de acciones localmente adecuadas puede generar problemas en la ejecución de las tareas globales. Es decir, el agente puede caer fácilmente en mínimos locales.

La aproximación oportunista es una estrategia intermedia. Ello hace que esta aproximación tenga los problemas y ventajas de ambas aproximaciones, pero tanto unos como otros suavizados en función cada implementación concreta. Cada una hace especial incapié en mitigar parte de los problemas (los más relevantes en su entorno principal de utilización), lo que en general conlleva el agravamiento de los restantes; y en intensificar las ventajas que resulten más interesantes (de nuevo en función del entorno para el que se diseña el sistema), haciendo en ocasiones que el resto de las ventajas desaparezca.

Una vez analizadas las alternativas principales para el control de robots, con sus pros y sus contras en los distintos tipos de entornos, resta realizar la caracterización inversa, esto es, analizar las características de los entornos de la robótica autónoma para decidir qué alternativa es la más apropiada en cada uno. La Tabla 2.2 pretende reflejar algunas de las dimensiones más relevantes de los dominios típicos de la robótica. Esta tabla clasifica (con + para la alternativa más adecuada, − para la menos y $\sim$ para la intermedia) las tres estrategias fundamentales presentadas en las secciones anteriores.

Tanto en los resúmenes anteriores de los problemas como en la caracterización de la Tabla 2.2, se consideran las ideas genéricas de cada estrategia de control, no las implementaciones particulares de las mismas. Cada una de las implementaciones

Característica	Simbólico	Reactivo	Oportunista
Incertidumbre en Operadores	−	+	∼
Incertidumbre en Información	−	+	∼
Metas Explícitas	+	−	∼
Representación del Estado	+	−	∼
Velocidad de Reacción	−	+	∼
Dinamismo	−	∼	+
Coste del Diseño	+	−	∼
Capacidad de Comunicación	∼	−	+
Coordinación	−	∼	+
Cooperación	+	−	+

Tabla 2.2: Caracterización de los sistemas de planificación

particulares, o las adaptaciones de sistemas generales, precisamente lo que intenta es limitar esos problemas. Por ejemplo, PRODIGY se ha adaptado de diversas formas: para considerar la existencia de múltiples agentes, (trabajo realizado por Alicia Pérez [Pérez, 1991]); para tratar la incertidumbre del entorno, (realizado por Jim Blythe [Blythe, 1994]), etc.

Las características analizadas se pueden clasificar, como indica la Tabla 2.2, en tres grupos: el referido a las características del entorno, las consideraciones temporales y las relativas al número de agentes involucrados. Las referidas al tipo de entorno incluyen tanto las características del entorno propiamente dicho (si está estructurado, el grado de dinamicidad del sistema, etc.) como las características en las que se basan la arquitecturas (incertidumbre en las informaciones, las operaciones etc.). Así, parece claro que en entornos altamente dinámicos, donde las suposiciones acerca del mundo no se cumplen, los planificadores tradicionales, que se basan en ellas, no son apropiados. Tendrían que estar continuamente replanificando sus acciones a medida que el mundo variase.

Por su parte, los sistemas reactivos son capaces de reaccionar sin problema ante este tipo de situaciones, pues las relaciones "estímulo-respuesta" están fijadas de antemano. Sin embargo, los sistemas reactivos, donde no hay ninguna representación abstracta del mundo ni de las metas, son muy difíciles de aplicar a tareas de alto nivel, que en general implican combinar operadores sencillos. La aproximación oportunista decide las acciones con una búsqueda muy reducida, lo que las hace ser casi tan eficientes como los reactivos, permitiendo sin embargo establecer combinaciones de comportamientos simples para conseguir acciones más complejas. La restricción es que dichos planes no

son tan flexibles como los de los sistemas simbólicos.

Otra característica importante que se debe considerar es el esfuerzo de diseño necesario para adaptar un sistema a cada entorno. Los sistemas simbólicos utilizan en general una representación de alto nivel, basada en operadores, predicados para la descripción del mundo, etc., que los hacen fácilmente adaptables. Los sistemas reactivos por contra se construyen como procesos dedicados diseñados "ad hoc" para cada aplicación específica, lo que implica que un sistema reactivo tiene que ser completamente reconstruido para cada aplicación. Los sistemas oportunistas vuelven a ser un caso intermedio. Los controles simples se diseñan para tareas específicas, como los reactivos, pero la forma en que se combinan puede ser fácilmente adaptada a los distintos dominios.

Por lo que respecta a los requisitos temporales, si la acción a realizar tiene que decidirse en muy corto espacio de tiempo, por ejemplo el giro ante un obstáculo, realizar una búsqueda muy pesada parece poco apropiado. Por este motivo, la mayoría de los planificadores tradicionales basados precisamente en este principio suelen descartarse en los entornos definidos como de tiempo-real. Por otra parte, si el tiempo no es un factor crítico, estos planificadores son capaces de encontrar mejores soluciones que los reactivos, que pueden no llegar a encontrar la solución en casos de mínimos locales. Por su parte, las aproximaciones oportunistas representan un compromiso entre ambas variantes. Por un lado permiten la toma aceptablemente rápida de decisiones, realizando una búsqueda muy reducida y por otra permiten establecer "planes", es decir, concatenación de acciones con un fin. El problema es que esos planes son menos flexibles que los de planificadores simbólicos.

La influencia que ejerce la existencia de varios agentes se puede evaluar estableciendo una gradación en el trabajo conjunto de los agentes. En un primer nivel estaría la "comunicación", donde los agentes únicamente intercambian información, lo que suele hacerse compartiendo un lenguaje, que a su vez sueler implicar el uso una representación interna del mundo. El segundo nivel es el de "coordinación", que se suele entender como la ejecución de las tareas de los robots sin interferencias. Para ello es necesario utilizar algún mecanismo de comunicación. Esta comunicación puede ser implícita, generalmente en sistemas de tipo reactivo; o explícita, utilizando un lenguaje. El tercer nivel es el que se suele calificar como "cooperación", e implica que las acciones de los agentes se organizan para maximizar el rendimiento del grupo. Este nivel exige poder establecer metas comunes, para lo que se necesita algún mecanismo de comunicación;

y procedimientos comunes, que exigen mecanismos de coordinación.

Contrastando la caracterización de la Tabla 2.2 con la del modelo que se quiere construir (Sección 1.2), se concluye que la estrategia más apropiada para el tipo de problemas que trata de resolver este trabajo es la planificación oportunista. Sin embargo, las arquitecturas oportunistas existentes (RAP, TR, InteRRaP, ...) no abordan ciertos requisitos muy importantes en los dominios de interés para este trabajo, como la existencia de múltiples agentes.

Por todo ello, se puede definir ABC^2 como un modelo para el control de agentes autónomos, basado en los principios de la planificación oportunista e influenciada por los sistemas multi-agente, diseñado especificamente para controlar robots autónomos en entornos poco estructurados y dinámicos.

Finalmente, indicar que no se han considerado en este análisis del estado del arte otros tipo de aproximaciones a la planificación y el control, como la planificación estocástica [García-Martínez and Borrajo, 1997], la planificación genética [Muslea, 1997], la basada en grafos [Blum and Furst, 1995], etc., pues son sistemas muy específicos y que hasta ahora no han tenido mucha influencia en el campo de los sistemas de control de robots autónomos, que es el objetivo de esta tesis. Tampoco se han tenido en cuenta los trabajos basados en el razonamiento basado en la duración de las acciones, más relacionado con el campo de los sistemas de tiempo real estricto o con el *scheduling* [Drummond *et al.*, 1994] que con la toma de decisiones, que es el aspecto central del modelo que se expone en el siguiente capítulo.

Capítulo 3

Definición del Modelo

Good research comes from thinking that something is fun, and playing with it.

Leslie Lamport

Los objetivos enunciados en la Sección 1.2 exponen la meta global de este trabajo: el diseño de un modelo general para la ejecución de tareas de alto nivel en un entorno no completamente estructurado, dinámico y donde pueda existir más de un agente. Se propone además como motivación fundamental demostrar que es posible integrar la reactividad con mecanismos de planificación simbólica. Por ello, la construcción de la arquitectura propuesta debe permitir integrar en el mismo modelo acciones implementadas según el paradigma más adecuado para cada tipo. Por ejemplo, usando lógica borrosa en controladores eficientes, redes neuronales en tareas de reconocimiento, etc. Además, el modelo debe facilitar la cooperación entre los agentes permitiendo que las acciones puedan ser decididas por el agente o pedidas por otros agentes que cooperen con él.

Este capítulo se dedica a formalizar el modelo desarrollado según estas restricciones, a explicar su funcionamiento y a compararlo con aquellos modelos que puedan tener alguna similitud. Para ello en la Sección 3.1 se describen brevemente las ideas básicas en las que se fundamenta este modelo. A continuación, en la Sección 3.2, se describen formalmente, detallando sus componentes en la Sección 3.3. En la Sección 3.4 se explican los algoritmos básicos necesarios para su funcionamiento. Por último, la Sección 3.5 compara el modelo con otros sistemas similares.

3.1 Fundamentos del Modelo

En el primer capítulo de esta memoria se exponían informalmente los objetivos de este trabajo, y se concretaban en la creación de un modelo que permitiese controlar un robot en un entorno dinámico, considerando que pueda existir más de un agente. Una vez repasado el estado del arte en este campo, este objetivo se puede describir más formal y concretamente como el diseño, construcción y prueba de un modelo de planificación de las acciones de un robot en un entorno dinámico, basándose para ello en el modelo oportunista descrito en [Hayes-Roth, 1992]. Es decir, en crear un sistema que sea capaz de tratar las contingencias de forma reactiva.

Además de la definición puramente científica, el modelo deberá cumplir ciertas restricciones "ingenieriles": que permita integrar diferentes modelos básicos de control, como el control de robots mediante reglas borrosas [Matellán *et al.*, 1995b]; y que sea configurable, es decir, que el uso del mismo modelo en distintos entornos no implique su rediseño completo; que sea escalable, esto es, que sea fácilmente ampliable y adaptable a problemas de diversa envergadura; etc. Algunas de estas restricciones han supuesto decisiones de diseño, que se comentan en este capítulo; y otras de implementación, que se comentan en el siguiente.

Con estos requisitos y los antecedentes expuestos, se ha diseñado un modelo para el control de robots autónomos, al que se ha denominado ABC^2, que basa su funcionamiento en el uso de comportamientos predefinidos de tipo reactivo que cada robot es capaz de componer de forma oportunista para obtener un comportamiento más complejo. Para realizar la composición se utiliza una agenda que mantiene una lista de acciones pendientes. Cada una de esas acciones puede necesitar (o no) de otras acciones predefinidas más simples. Las acciones pueden ser el resultado de metas propias o de peticiones de otros robots. Para ello se establece un mecanismo de comunicación entre los robots, que se realiza mediante el procedimiento de todos a todos (*broadcast*).

En cuanto al diseño de las acciones básicas, se permite el uso de diferentes mecanismos, que se pueden elegir de forma independiente para cada una de ellas. Es decir, una acción podrá realizarse mediante un controlador borroso, otra puede ser el resultado de un comportamiento aprendido, etc. y todas ser parte de las habilidades del mismo agente. El modelo también tiene en cuenta que las habilidades de los robots, es decir, las acciones que son capaces de realizar, pueden ser diseñadas por humanos, por lo que deberá presentar un mecanismo de diseño fácilmente asimilable por cualquier

programador.

3.2 Formalización del Modelo

El modelo que se ha diseñado intenta describir con su nombre, ABC^2 (**A**genda **B**ased **C**ooperation for **A**gent's **B**ehaviors **C**oordination), sus características, que consisten en la realización un sistema que sea capaz de combinar controladores especializados en una actividad determinada, que se denominarán en adelante *habilidades*, para conseguir un comportamiento más complejo. Estas habilidades básicas tienen un carácter marcadamente reactivo.

Cada habilidad que el robot, o en general el agente, decida ejercer (ejecutar, utilizando jerga informática) puede necesitar el uso de otras habilidades que hagan posible su ejecución. En ese sentido, se trata de un concepto similar al de las pre-condiciones en la planificación simbólica clásica.

La lista de habilidades H_i que pueden facilitar la ejecución de una habilidad dada H se denomina en lo sucesivo *lista de necesidades* y a cada elemento H_i se le considera una necesidad de H. A su vez, cada H_i puede tener su propia lista de necesidades. En esta versión de ABC^2 no se ha previsto ningún mecanismo formal para la detección de ciclos en la especificación de la listas de necesidades, siendo responsabilidad del diseñador el control de los mismos.

El sistema no decide de forma directa la ejecución de las habilidades, sino que éstas se insertan como contenido de una estructura con un nivel de abstracción mayor que se denominará en adelante *acto*. La combinación de las habilidades se realiza por medio de una agenda que organiza la ejecución de los actos que el agente considera en un determinado momento. La clasificación de los actos empleados en ABC^2 es similar a la empleada en CooperA [Sommaruga, 1993] , constituyendo una abstracción común a muchos sistemas multiagente [Bond and Gasser, 1988]. Se pueden resumir los tipos de actos en:

Actos de ejecución: representan la intención de ejecutar una acción. La intención de ejecutar esa acción puede deberse a la propia iniciativa del agente o ser el resultado de una petición de otro.

Actos de información: sirven para intercambiar información con otros agentes. El contenido de esa información puede ser por ejemplo su situación, el estado de ejecución de una tarea, información sobre el entorno, etc.

Actos de negociación: sirven para establecer la cooperación entre agentes, gestionando la petición, adecuación y aceptación o rechazo de tareas.

Los actos de ejecución pueden incorporarse a la agenda por diversos motivos:

- Se puede establecer un conjunto básico de acciones que el agente debe realizar (similar al concepto de meta en un sistema de planificación simbólica).
- Pueden insertarse si son necesarios para posibilitar la ejecución de otros, es decir, si la habilidad que contengan está en la lista de necesidades de una acción que desea ejecutarse.
- Pueden generarse como resultado de acontecimientos, eventos, producidos en el mundo.
- O por peticiones directas de otros agentes.

De igual forma, los actos pueden conseguir sus objetivos de diversas maneras:

- Mediante su ejecución.
- Por ejecución de otras acciones que como resultado colateral proporcionen los efectos deseados.
- Por acciones de otros agentes.
- Por simples modificaciones casuales en el entorno.

Los tres últimos casos son ejemplos de lo que se considera ejecución ocasional u oportunista de acciones.

Un agente, definido según ABC^2 , se compone de una estructura formada por un conjunto de atributos, estáticos unos y dinámicos otros. Entre los estáticos están:

- El nombre que tendrá el agente en el mundo (N), y que será por el que le conocerán el resto de los agentes.
- La lista de sus habilidades (H).
- La información sobre el nombre del resto de los agentes, que incluye su nombre y la lista de habilidades de ese agente. A esta información se le ha denominado *páginas amarillas* (P).

- Las informaciones que el agente va a conocer del mundo, que se definirán mediante un lenguaje (L).
- Un conjunto de heurísticas de control (C) que condicionarán su comportamiento.

En la primera implementación del modelo que se describe en esta tesis, no se ha considerado la posibilidad de modificar dinámicamente estos atributos, lo cual constituye claramente una línea de trabajo futuro (ver Sección 7.2).

Con estas definiciones, un agente (A) en ABC^2 se define mediante una tupla:

$$A =< N, H, P, L, C >$$

Con el mismo formalismo, un conjunto de agentes, que estará formado al menos por un agente, se representa como $< N, H, P, L, C >^+$.

Por ejemplo, la definición de los componentes estáticos del primer robot de un grupo en el que solo hay dos robots (`robot_1, y robot_2`) sería:

$$\begin{aligned}\texttt{robot\_1} = < \; & robot_1, \\ & \{habilidad_{11}, habilidad_{12}, \ldots, habilidad_{1n}\}, \\ & \{robot_2\text{: } habilidad_{21}, habilidad_{22}, \ldots habilidad_{2m}\}, \\ & \{Info\_Estructurada^+, Info\_No\_Estructurada^+\}, \\ & < heurística_{11}, heurística_{12}, \ldots heurística_{1k} > \\ >\end{aligned}$$

donde $habilidad_{1i}$ representa cada una de las habilidades de las que dispone el robot. Por ejemplo, la habilidad $habilidad_{11}$ podría ser `Ir_Bola` en un robot definido para el entorno del fútbol entre robots descrito en el Capítulo 6. A continuación se encuentra la definición de las habilidades de los demás robots. En este caso las del `robot_2`. Le sigue la descripción del entorno, para lo que se utiliza el lenguaje L. Esta información se organiza como *información estructurada*, compuesta por una jerarquía tradicional de conceptos, como por ejemplo `Bola` y sus atributos (dirección, distancia, etc.); y por *información no-estructurada*, formada por aquellos conocimientos que no caben en jerarquías como la anterior, como por ejemplo si el equipo está atacando o defendiendo, la situación del juego, etc. Por último, la definición de un robot debe incluir las heurísticas que controlarán el funcionamiento de la agenda.

La información dinámica de un agente define la situación en la que se encuentra dicho agente. Esto quiere decir que estará compuesta por una serie de elementos

cuyos contenidos irán variando a medida que transcurra el tiempo. La primera y más importante de estas estructuras es la agenda (A_g). Contiene los actos que el agente está considerando en un determinado momento. Las comunicaciones (C) por su parte representarán el estado del modulo encargado del intercambio de datos con el resto de agentes, esto es, los mensajes recibidos pero que no han sido procesados y los mensajes pendientes de envío. Por último, el agente irá actualizando la información (I) sobre el entorno que le rodea definida de acuerdo al lenguaje (L).

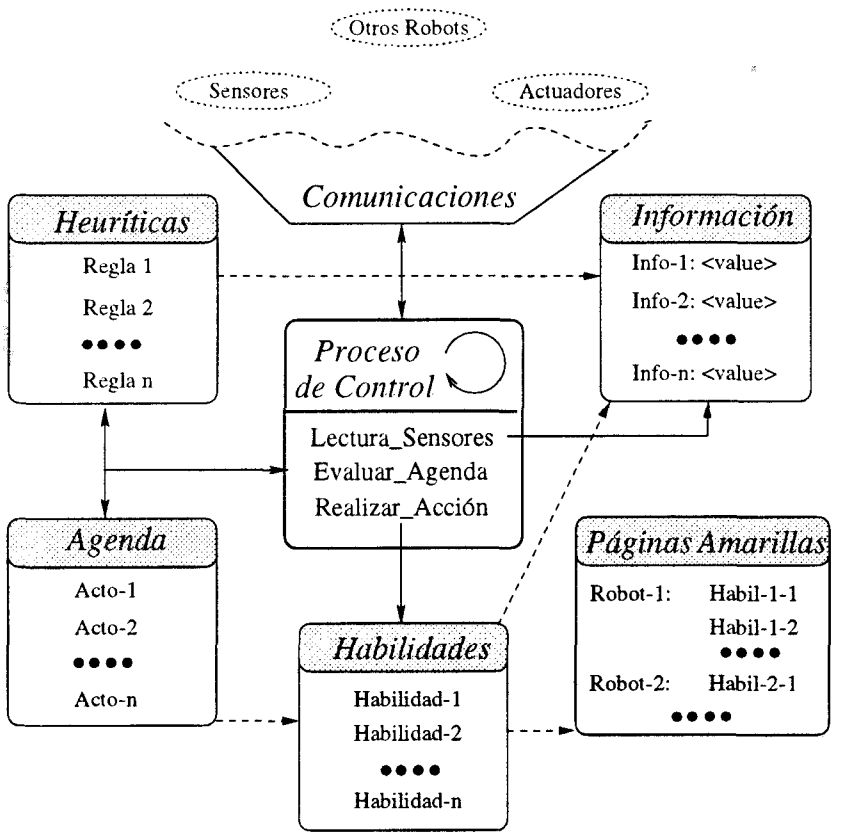

Figura 3.1: Modelización de un robot en ABC^2.

De esta forma, un agente de ABC^2 en un momento determinado se define como $Agente =< A_g, I, C_m >$, y la situación del grupo de agentes (por ejemplo, el conjunto de robots que forma un equipo de robo-fútbol) vendrá dada por $< A, A_g, I, C_m >^+$, donde A representa la información estática de cada agente (nombre, habilidades, páginas amarillas y heurísticas). La representación gráfica del modelo completo se presenta en la Figura 3.1.

En el centro de la figura se representa el proceso central de control del agente. En el caso de un agente para el control de un robot autónomo (el representado en la figura) este proceso tendría una estructura similar a la de los sistemas simbólicos

(Figura 1.1). Es decir, lectura de los sensores, búsqueda de la habilidad a ejecutar y ejución de la misma. En la figura (parte superior) se ha representado una abstracción, denominada *Comunicaciones* que se encarga de aislar al agente de los problemas de comunicación con sus entradas (sensores en el caso de un robot) y salidas (actuadores en un robot). De tal forma que, por ejemplo en un robot autónomo , el proceso de lectura de sensores consiste simplemente en actualizar las informaciones, encargándonse el módulo de comunicaciones del pre-procesamiento de la información.

En el caso de ABC^2 la búsqueda de la habilidad a ejecutar consiste en elegir uno de los actos contenidos en la agenda, para lo cual se emplean las heurísticas. A su vez cada uno de dichos actos contendrá alguna de las habilidades del robot, o de algún otro robot. En la figura se representan separadamente, bajo el epígrafe *Habilidades* se representan las habilidades propias del agente y en *Páginas Amarillas* las correspondientes a los otros robots. La ejecución de una habilidad puede requerir consultar alguna de las informaciones almacenadas utilizando el lenguaje L. De la misma forma, las heurísticas pueden utilizar cualquiera de las informaciones para decidir qué acto se ejecutará.

Concretando la definición formal del ejemplo anterior, una posible situación en un instante dado de dos robots que formen parte de un mismo equipo de fútbol [1] podría definirse como:

$< robot_1, \quad A_g = \{$ [HACER:Ir_Bola] $\},$
$I = \{$(Bola.distancia= $2m.$, Bola.ángulo = 90^o), (Defendiendo = $True$)$\},$
$C_m =$ <[PEDIDO:Ir_Bola, $robot_2$] >
>
$< robot_2, \quad A_g = \{$[HACER:Ir_Posición][PEDIR:$robot_1$Pasar]$\},$
$I = \{$(Bola.distancia= $50m.$, Bola.ángulo = 35^o), (Portería.distancia= $10m$)$\},$
$C_m =$ <[vacía]>
>

que indica que `robot_1` tiene un solo acto en su agenda. Dicho acto indica que el robot se plantea ejercer la $habilidad_{11}$ enunciada anteriormente, es decir, ir hacia la bola. Según esta definición el `robot_1` sólo tiene información estructurada acerca de un elemento del mundo que es la bola. Sobre dicho concepto sólo conoce información acerca de los atributos distancia y ángulo. Finalmente, la cola de mensajes contiene un mensaje recibido del `robot_2` en el que pide al `robot_1` que ejercite su habilidad `Ir_Bola`.

[1] Este dominio se analizará en detalle en el Capítulo 6.

De igual forma el `robot_2` tiene dos actos en su agenda, uno de ejecución y otro de negociación; dispone de información estructurada acerca de la bola, así como no estructurada sobre su modo de juego (este caso para indicar que está defendiendo).

3.3 Descripción de los Componentes

Una vez presentados los fundamentos de ABC^2 y enumerados sus elementos describamos con más detalle cada uno de los componentes reflejados en la Figura 3.1:

Habilidades. Conjunto de controladores reactivos sencillos que implementan comportamientos predefinidos que cada agente puede realizar por sí mismo. Se pueden implementar usando cualquier mecanismo de toma de decisiones, bien generado por un programador o mediante un método de aprendizaje automático. Por ejemplo, en esta memoria se describen habilidades construidas como controladores borrosos diseñados heurísticamente (ver Sección 5.2.1), esto es, el programador escribe las reglas borrosas de control. Otros comportamientos se han generado mediante aprendizaje genético (ver Sección 5.2.2); y otros como simples procesos de decisión (ver Sección 6.2.1).

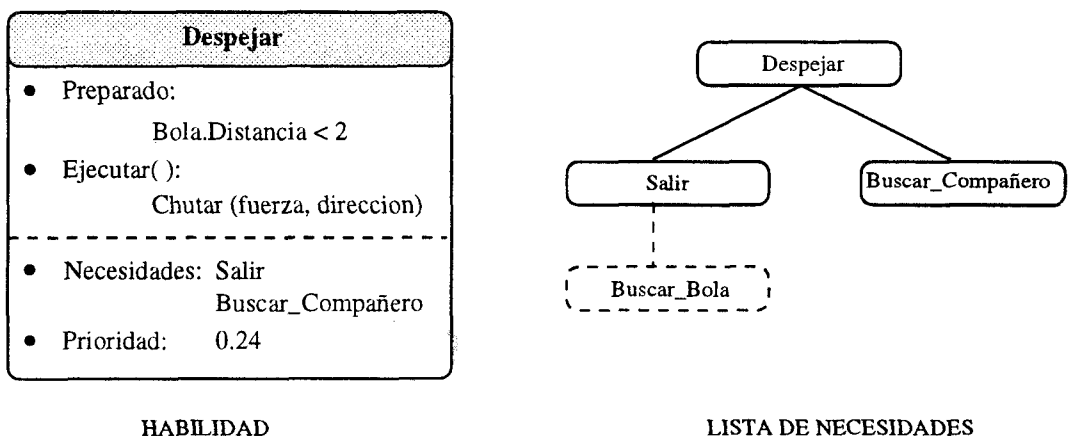

Figura 3.2: Componentes de una habilidad.

La definición de una **habilidad** particular, como la que se muestra en la Figura 3.2, que correspondería a la capacidad de despejar la bola, en el dominio del fútbol, consiste en definir:

- La condición por la cual se dispara el controlador se denomina *Preparado*. Esta condición debe devolver un valor booleano que indica si la acción que implementa la habilidad se puede ejecutar o no. En ese sentido se puede

considerar que es equivalente a verificar que las precondiciones de un operador de un planificador tradicional son ciertas. O bien, la condición de activación de un controlador de tipo reactivo.

La condición *Preparado* puede definirse sobre variables de tipos diversos. En particular, pueden usarse variables definidas sobre la información que el agente mantiene sobre su entorno, sobre valores obtenidos directamente sobre los sensores, se pueden utilizar parámetros de la configuración, etc.

- Un controlador que realice la acción propiamente dicha. Este controlador se ha representado en la Figura 3.2 en forma de función denominada *Ejecutar*. Aunque su interpretación debe ser equivalente a un comportamiento básico en una arquitectura reactiva o a las post-condiciones (efectos) de un operador de un planificador tradicional.

  La forma en que se implemente este controlador es libre pudiendo ser, desde una simple asignación de valores predeterminados a los parámetros de control de los actuadores, hasta una costosa búsqueda de dichos valores mediante un planificador tradicional.

- La lista de necesidades. Esto es, una lista de habilidades que puedan hacer que ésta sea "ejecutable", o lo que es lo mismo, que puedan hacer que la condición *Preparado* pase de ser `FALSA` a ser `VERDADERA`. La composición de las habilidades de un robot daría lugar a un conjunto de árboles como el que aparece en la parte derecha de la Figura 3.2, donde se refleja la lista de necesidades de la habilidad *Despejar* junto con la de la habilidad *Salir*

  La utilidad de esta lista es reducir el proceso de búsqueda en un espacio de estados de crecimiento exponencial que se produce en los planificadores clásicos. Esta lista sustituye el proceso de búsqueda por la simple inclusión de la lista de necesidades en la agenda.

- La *Prioridad* de la habilidad. Este valor de prioridad es un parámetro heurístico "a priori" que indica la importancia de la habilidad. Las heurísticas pueden utilizar este valor como un criterio más a la hora de ordenar los actos de ejecución.

El ejemplo de la Figura 3.2 muestra los componentes de la habilidad `Despejar` del problema del fútbol entre robots. El ejemplo resume la definición de una habilidad que hiciese a un portero despejar la bola. Un ejemplo que utiliza la

versión real de esta habilidad se presenta en el Anexo B.

Se ha dividido la representación de la habilidad en dos partes. La superior muestra la condición para su ejecución (*Preparado*) y el propio controlador (*Ejecutar*). Siguiendo con el ejemplo, la condición para que el agente pueda despejar (chutar la bola con la máxima potencia), es que tenga la bola lo suficientemente cerca como para poder hacerlo. En el dominio del fútbol entre robots esa distancia es 2. La ejecución se ha simplificado a la invocación de una función que implementa la acción de golpear la bola con la fuerza y dirección especificadas.

La parte inferior está formada por informaciones, concretamente la prioridad inicial asignada a la habilidad y la lista de necesidades que pueden hacer que se cumpla la condición *Preparado*, como por ejemplo *Salir* de la portería en busca de la bola. Las habilidades de esta lista son a su vez otras habilidades.

La Agenda y los Actos. La *Agenda* es una estructura dinámica que contiene actos. El nombre de estos elementos se debe a la teoría de los "Actos de Comunicación" (Speech Acts Theory) [Cohen and Perrault, 1986]. Su utilidad principal es proporcionar un nivel de abstracción mayor que el de las habilidades de forma que los agentes los puedan utilizar para cooperar.

El modelo ABC^2 considera una versión reducida del conjunto de posibles actos de comunicación. En concreto se han implementado cinco actos diferentes, a saber:

`HACER <habilidad>`, que representa las habilidades potenciales que un agente puede realizar por sí mismo.

`PEDIR <agente> <habilidad>`, para solicitar a otro agente que realice una tarea particular.

`PEDIDO <habilidad> <agente>`, para indicar que el agente indicado ha pedido, al que esté evaluando este acto, la ejecución de la habilidad que se indica en el argumento.

`INFORMAR <información> <agente>`, significa que la información dada en el argumento debe ser enviada al agente indicado.

`INFORMADO <información> <agente>`. Es el acto correspondiente al anterior. Es decir, es el acto que se genera en un agente que recibe una información.

La Figura 3.3 representa un posible conjunto de actos en la agenda. Sus componentes son:

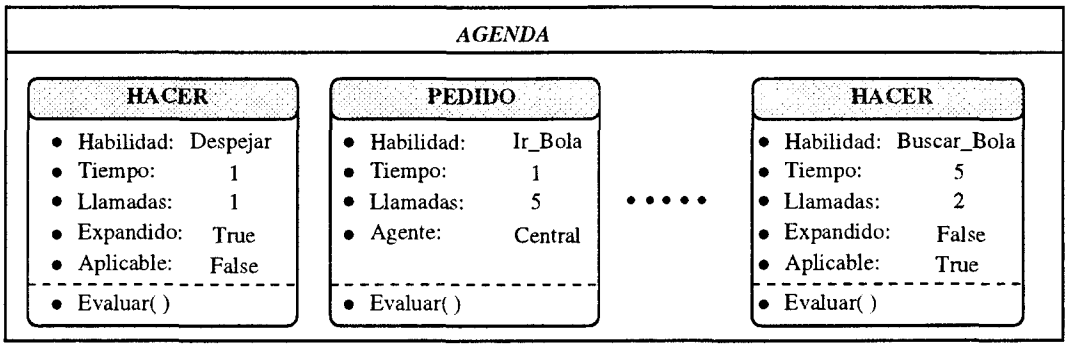

Figura 3.3: Componentes de un acto dentro de la agenda

- El tipo de acto (`HACER`, `PEDIDO`, etc.);
- La habilidad asociado al acto, por ejemplo la habilidad a realizar en el caso de actos `HACER`.
- El atributo `Llamadas` que indica la cantidad de actos también presentes en la agenda que han solicitado la ejecución de éste.
- El atributo `Tiempo` que informa del instante en que este acto fue insertado en la agenda.
- El atributo `Expandido` que indica, en un acto de ejecución, si las necesidades de la habilidad contenida en dicho acto se han introducido en la agenda o no. Este atributo no existe en los demás tipos de acto.
- El atributo `Agente` representa el agente con el que establece la comunicación en los actos de información y negociación, no existiendo en los de ejecución.
- El atributo `Aplicable` que indica si el acto se puede evaluar o no. Por ejemplo, en el caso de actos de ejecución depende de la condición de activación de la habilidad a realizar.
- La función `Evaluar`, que indica qué hay que hacer con ese acto cuando se desee evaluarlo. Por ejemplo, para un acto del tipo `HACER` será ejecutar la habilidad correspondiente; para uno del tipo `INFORMAR` enviar el mensaje, etc.

La Sección 3.4 presenta en más detalle cómo la agenda utiliza los atributos de los actos que contiene para decidir las acciones del agente.

Heurísticas. Influyen en la decisión del acto que se seleccionará de la agenda para su ejecución. El diseño actual considera la ejecución exclusiva de los actos. Es decir,

en un instante determinado el agente realizará uno y sólo uno de los actos de la agenda. Esto es así porque se ha considerado que este modelo deberá encargarse de gestionar las tareas de alto nivel, dejando a otro nivel del software la gestión de las tareas periódicas como leer sensores, monitorizar los controladores de bajo nivel de los motores de los motores, PIDs , etc.

Las heurísticas se pueden definir en función de diferentes factores, como por ejemplo:

- La prioridad inicial de la habidad asociada al acto de ejecución.
- El tiempo que el acto ha permanecido en la agenda.
- El número de actos que han solicitado su ejecución.
- La información del entorno.
- La definición del agente.

El resultado de la evaluación de las heurísticas será una ordenación de los actos contenidos en la agenda. Por ejemplo, se podrían definir las heurísticas de control en forma de reglas borrosas. Las variables de entrada de estas reglas representarían los factores anteriores y la salida que proporcionaría este sistema de reglas sería un "peso" o prioridad para cada uno de los actos contenidos en la agenda. Una vez asignados, la agenda simplemente debe ejecutar el acto de mayor peso. Una regla heurística de este tipo tendría la siguiente forma:

Si *Atacando* **Es** Verdad **Y** *Distancia(Bola)* **Es** Cerca **Entonces**
`Ir_Bola.`*Prioridad* **Es** *Aumentar_Mucho* **Y** `Ir_Posicion.`*Prioridad* **Es** *Disminuir*

Por otra parte, la literatura centrada en los sistemas multi-agentes suele distinguir dos partes dentro de las heurísticas de control de los agentes. Así, distinguen entre las heurísticas básicas de elección de la acción más prioritaria y un conjunto de reglas específico para tratar los actos de negociación. Estas heurísticas de negociación especifican cuál será el comportamiento del agente respecto a sus compañeros. Esto es, si será un agente colaborador o por el contrario egoísta. Dicho de forma más específica, si aceptará frecuentemente ejecutar acciones pedidas por sus compañeros (comportamiento altruísta), o si por el contrario las rechazará sistemáticamente (egoísta) [Sommaruga, 1993, Molina, 1997]. Estos

planteamientos tienen mucho que ver con la psicología y la sociología y han sido poco considerados en la implementación del sistema.

En cualquier caso, la definición de las heurísticas de ABC^2 que se acaba de exponer permite integrar sin ninguna dificultad ese tipo de heurísticas de cooperación. Para ello basta con añadir reglas cuyas variables de los antecedentes estén definidas en función de la información de cooperación.

Páginas Amarillas. Almacenan la información que un agente tiene sobre las habilidades de otros agentes. Esta información se almacena en forma de tabla accesible por habilidad. Esto es, un agente puede obtener información sobre qué agentes son capaces de realizar cierta hablidad.

En principio no se establece cual es el mecanismo de generación de esta tabla. Puede ser un conocimiento "a priori" del agente o ser un conocimiento adquirido mediante actos de comunicación.

En el modelo tampoco se han establecido restricciones respecto al uso de las habilidades conocidas de otros agentes, pudiendo emplearse éstas como habilidades propias. Esto es, el agente puede por ejemplo incluirlas en las listas de necesidades de sus habilidades. En ese sentido, las habilidades de un agente se pueden considerar como abstracciones de las acciones del agente para hacerlas accesibles a otros miembros del equipo. De hecho, esto equivale a decir que el agente tiene cierto meta-conocimiento sobre sí mismo, dado por la defincón de sus habilidades, y sobre sus compañeros, a través de la información contenida en las páginas amarillas.

Información. Se refiere al modelo que mantiene el agente acerca del mundo en el que trabaja. La variedad de entornos en los que se puede desenvolver un agente hace inviable la creación de una representación genérica para la información. Por ejemplo, no tiene que ver la información recibida del simulador de la RoboCup, que es de tipo lingüístico, con los valores numéricos leídos de los sensores de un robot real. En lugar de la representación genérica se ha utilizado un lenguaje reducido para manejar la información.

Esta información se organiza como *información estructurada* e *información no-estructurada*. La primera está compuesta por una jerarquía tradicional de conceptos relevantes para la tarea del agente y sus atributos. Por ejemplo, la infor-

mación acerca de la bola se estructura en función de atributos como la dirección en que se encuentra, la distacia, etc. La segunda está formada por aquellos conocimientos que no encajan claramente en la jerarquía específica del dominio. Por ejemplo, información referida a sí mismo — como la fiabilidad de los sensores o a la situación en general—, a los compañeros, etc.

Atendiendo a este criterio, este modelo podría clasificarse dentro de los sistemas cognitivos, es decir, aquellos que mantienen un modelo interno del mundo, mientras que en los modelos basados en arquitecturas reactivas puras las salidas para los actuadores se calculan directamente como función de los sensores.

Comunicación. Es una de las capacidades distintivas de los agentes diseñados siguiendo el modelo ABC^2. Por ello se ha definido un módulo específico que se encargue de gestionar la complejidad intrínseca de las comunicaciones (protocolos, colas de mensajes, etc.).

La abstracción de las comunicaciones constituye el nivel más bajo del mecanismo de cooperación, el cual se basa en la teoría de los actos de comunicación previamente descrita. La abstracción implementada en esta versión ofrece una interfaz única para el envío de mensajes basada en la llamada:

```
send (Mensaje, Destino)
```

Este módulo proporcionará el nivel de abstracción necesario para manejar los distintos entornos de comunicación. La forma en la que se implemente dependerá del tipo de sistema de comunicaciones disponible en el entorno de trabajo del agente. Cuando se consideren nuevas versiones de la arquitectura sería interesante utilizar abstracciones más complejas que puedan utilizar toda la potencia de los protocolos de comunicación grupal [Barahona, 1998].

Como ejemplo, en un simulador, la función se podría implementar sobre la torre de protocolos TCP/IP disponible hoy en día en cualquier ordenador, mientras que en un robot real habría que utilizar el sistema de comunicación disponible. Por ejemplo, en el simulador de la RoboCup [Itsuki, 1995], se han empleado sockets UDP, mientras que en los experimentos descritos en la Sección 5.3 la comunicación entre dos robots se ha establecido usando el puerto serie.

3.4 Ciclo de Control

La sección anterior describe los componentes de un agente en ABC^2. La unión de todos ellos resulta en la imagen del agente en un instante dado. La forma en que varía el contenido de las estructuras de datos enumeradas en el apartado anterior, que combina los distintos elementos, los usa y modifica y que en definitiva corresponde al control del agente, se denominó "Proceso de Control" en la Figura 3.1. Esta sección esta dedicada a formalizar los algoritmos que realizan esa función.

Así, el algoritmo correspondiente al ciclo básico de control (Proceso de Control) se puede resumir en la Formalización 3.1.

Formalización 3.1 Ciclo básico de control

Inicializar ($Agenda$);
Mientras $Agenda \neq \emptyset$
  Repetir $\forall Acto_i \in Agenda$
    Si $Acto_i.Llamadas = 0$ **Entonces** Borrar($Acto_i$)
  Hasta $Agenda$ no varíe
  $Actos_{aplicables} = \{Actos_i \in Agenda$ tales que $(Acto_i.Habilidad)$.Preparado = True $\}$
  Repetir $\forall Acto_i \notin Actos_{aplicables}$
    Si $(Acto_i.Habilidad)$.Expandido = $False$ **Entonces** Expandir($Acto_i.Habilidad$)
  Fin Repetir
  $Acto \Longleftarrow$ Aplicar_Heuristicas ($Actos_{aplicables}$)
  Evaluar ($Acto$)
Fin Mientras

donde: $Agenda$ es la agenda del robot, $Actos_{aplicables}$ es el subconjunto de los actos de ejecución contenidos en la agenda cuya habilidad asociada está *Preparada*. Es decir, las habilidades que el robot está considerando ejecutar y que se pueden ejecutar en ese instante porque el resultado de la condición de ejecución es cierta. `Inicializar` inserta el acto inicial en la $Agenda$, `Borrar` se encarga de eliminar un acto de la agenda, y disminuir el contador de llamadas de los actos, `Expandir` de introducir su lista de necesidades de una habilidad en la $Agenda$, `Aplicar_Heurísticas`, como su propio nombre indica, de calcular los pesos de los distintos actos y `Evaluar` de ejecutar un acto.

La forma en que funciona este algoritmo es la siguiente. En primer lugar hay que inicializar el agente. Esto se realiza insertando un acto de ejecución inicial en la agenda. Este acto se considera la meta principal o final del agente en ese entorno. El resto de

los actos se irán generando dinámicamente, por ejemplo, como necesidades de éste, por eventos en el entorno, porque lo indique la ejecución directa de la función *Ejecutar* de otra habilidad, etc.

La existencia de ese acto de ejecución inicial tiene justificaciones profundas que se pueden relacionar con conceptos abstractos como la "motivación". Es decir, si se considera un sistema autónomo, ¿cuál es el instante y en qué condiciones empieza a "existir" ese sistema? ¿cuál es el objetivo que tendrá ese agente y cómo se le especifica? ¿qué influencia tiene esa definición en la autonomía real del sistema? etc. La mayoría de esas preguntas quedan fuera del dominio de este trabajo, pero no por eso dejan de ser interesantes.

En lo que concierne a este trabajo, el acto inicial se entiende como la acción que marcará el final de la ejecución del agente. En el dominio de la RoboCup, por ejemplo, el acto inicial será el encargado de ejecutar la habilidad denominada `Ganar_Partido`. La condición de activación de esta habilidad será que el tiempo del partido haya terminado, es decir, la función *Preparado* sólo depende del tiempo transcurrido. La acción que realiza `Ejecutar` es mostrar el resultado del encuentro. La lista de necesidades de esta habilidad dependerá ya del tipo de jugador de que se trate. Un delantero tendrá, por ejemplo, habilidades del tipo `Marcar_Gol`, `Pasar`, etc. En el primer ciclo de control, como el tiempo no estará cumplido, se insertarán las necesidades de ganar partido y serán esos actos los que vayan marcando las acciones del agente en función de la situación en el campo.

Se podría haber considerado otra solución, quizá más eficiente para gestionar el "nacimiento" y la "muerte" de los agentes, pero habría sido menos consistente con el modelo. En cualquier caso, si en algún momento se considerase necesario, bastaría con modificar la condición que genera el final de la ejecución — ahora cuando la *Agenda* está vacía — por otra condición diferente.

El funcionamiento del ciclo de control es síncrono, dependiendo de las características del sistema sobre el que trabaje. Por ejemplo, en el simulador de la RoboCup un jugador puede enviar, como máximo, un comando cada 200 mili-segundos (dependiendo del modo de visión: ver Sección 6.1.2). Ese debería ser, por tanto, el ciclo de reloj en ese dominio.

En cada uno de esos instantes, el funcionamiento del sistema de decisión elige en primer lugar los *actos aplicables*. Para ello se consulta el atributo *Aplicable* del acto. Este atributo se calcula dependiendo del tipo de acto. Por ejemplo, en los actos de

ejecución corresponde al valor de la función *Preparado* de la habilidad contenida en el acto. Para otros tipos de actos se calcula en función de los parámetros de comunicación, como el tiempo desde que se envió el último mensaje en el caso de actos de información o si se aceptan peticiones si se trata de un acto como PEDIDO.

Formalización 3.2 Inserción de las necesidades de una habilidad

Expandir (H: Habilidad) **es**
  Repetir $\forall Habilidad_i \in H$.Lista_Necesidades
    Si $\exists Acto_j \subset Agenda$ tal que $Acto_j.Habilidad = Habilidad_i$ **Entonces**
      $Acto_j.Llamadas = Acto_j.Llamadas + 1$
    En otro caso
      $Agenda = Agenda+$ **nuevo** $Acto_k$(HACER: $Habilidad_i$)
      $Acto_k.Llamadas = 1$
    Fin Si
  Fin Repetir
Fin Expandir

Si un acto HACER no es aplicable, se comprueba el valor del atributo *Expandido*; si es falso, se invoca a la función Expandir que consiste en insertar las habilidades de la lista de necesidades en la *Agenda* en forma de actos [HACER: <Habilidad>] como muestra la Formalización 3.2.

Antes de insertar cada nuevo acto se comprueba que no estuviera ya en la agenda. Si ese acto ya estaba en la agenda, entonces no se incluye de nuevo, sino que se incrementa el contador Llamadas. Si no estaba, se incluye el nuevo acto en la agenda con el contador inicializado a 1, como refleja el pseudo-código anterior.

Al mismo tiempo que se seleccionan los actos aplicables, se eliminan de la agenda aquellos actos cuyo contador de llamadas se haya hecho cero. El mecanismo para realizar dicha función se muestra en la Formalización 3.3.

El hecho de que el contador *Llamadas* esté a cero indica que dicho acto no es requerido por ningún otro, por lo cual no tiene sentido que permanezca en la agenda. También debe retirar su petición de cuantos actos haya creado para expandir su lista de necesidades (si la expandió). Esto es, debe decrementar en uno el contador *Llamadas* de los actos HACER que incluyan habilidades de su lista de necesidades, para indicar que ya no los necesita.

Este algoritmo se aplica hasta que no se producen variaciones ni en el número de actos en la agenda, ni en los valores de los contadores *Llamadas*. También se

Formalización 3.3 Eliminación de actos de la agenda

Borrar (A: Acto) **es**
  Repetir $\forall Habilidad_i \in (A.Habilidad).Lista\_Necesidades$
    Si $\exists Acto_j \in Agenda$ tal que $Acto_j.Habilidad = Habilidad_i$**Entonces**
      $Acto_j$.Llamadas = $Acto_j$.Llamadas - 1
    Fin Si
    Si $Acto_j$.Llamadas = 0 **Entonces**
      Borrar($Acto_j$)
    Fin Si
  Fin Repetir
  $Agenda \longleftarrow (Agenda - A)$
Fin Borrar

decrementa el contador de llamadas de los actos que contienen habilidades de la lista de necesidades cuando se consigue ejecutar el acto. La justificación es obvia: si se ha conseguido ejecutar el acto significa que probablemente alguna de las necesidades lo ha facilitado, o bien, que el propio entorno ha hecho posible la ejecución. En ambos casos esos actos ya no son necesarios y por tanto se decrementa su contador de *Llamadas*.

Una vez extraídos los actos aplicables, se aplican las heurísticas del dominio a dicho conjunto de actos para seleccionar el que se ejecutará. Una vez elegido, se evalúa dicho acto. La evaluación de un acto puede verse en la Formalización 3.4.

Formalización 3.4 Evaluación de los actos de la agenda

Evaluar (A: Acto) **es**
  A.Llamadas = A.Llamadas -1
  Caso Tipo_de(A) **sea**
    $HACER$: Ejecutar ($A.Habilidad$)
    $PEDIR$: **send** ($A.Habilidad$, $A.Agente_i$)
    $PEDIDO$: $Agenda = Agenda+$ **nuevo** Acto($Hacer : A.Habilidad$)
    $INFORMAR$: **send** ($A.Info_i$, $A.Agente_j$)
    $INFORMADO$: Modificar($A.Info_j$)
  Fin Caso;
end Evaluar

Este es el algoritmo genérico de evaluación de un acto dependiendo del tipo de acto que sea. Así, si el acto a evaluar es de tipo `HACER`, la evaluación corresponderá a la ejecución de la habilidad contenida en el acto. Si es de tipo `PEDIDO` y se decide aceptar

la petición, su ejecución resultará en la inserción de un nuevo acto HACER en la agenda (como refleja la Formalización 3.4). Dicho acto contendrá la habilidad pedida. De la misma forma cada tipo de acto tendrá su evaluación particular.

El tratamiento de los actos de cooperación se realiza mediante los siguientes pasos:

1. Analizar la cola de mensajes recibidos. Dicha cola, que está gestionada por el módulo de comunicaciones anteriormente citado, consiste en una lista ordenada por instante de llegada de mensajes que no han sido analizados.

2. Comprobar, para cada mensaje, si el emisor está en el conjunto de agentes del que se aceptan peticiones e informaciones. La organización de la lista de agentes y relaciones que se aceptan de ellos dependerá del tipo de entorno. En algunos casos podrá ser una lista única, mientras que en otros se separará en dos para distinguir agentes que pueden proporcionar información y los que pueden hacer peticiones. Esta comprobación incluye mecanismos de seguridad [Ribagorda *et al.*, 1996] en ciertos dominios, como la RoboCup (ver Sección 6.1.2), donde pueden existir intentos de engañar a un agente.

3. Extraer los mensajes de información de entre los aceptados y actualizar la información ($Info_j$) contenida en ellos. Es decir, no se incluyen en la agenda, sino que se tratan directamente.

4. Introducir el resto de los mensajes, que serán peticiones, como actos [PEDIDO: $Habilidad_{pedida}$] en la agenda. El contador *Llamadas* se coloca a uno.

5. Evaluar la agenda según el algoritmo anterior (Formalización 3.4), con la salvedad de que la evaluación de la agenda realizará un paso previo a la comprobación de actos que no tienen llamadas. Ese paso consiste en convertir directamente los actos [PEDIDO: $Habilidad_i$] en [HACER: Hab_i] y en enviar los mensajes correspondientes a los INFORMAR y PEDIR.

De nuevo se ha mantenido el paso por la agenda, que puede parecer poco útil, para mostrar la coherencia del modelo. Si ahora se añadiese un mecanismo de negociación de las tareas basado en algoritmos como la red de contratos [Smith, 1980], bastaría con añadir las heurísticas adecuadas, añadir al análisis de los mensajes de información semántica sobre la aceptación y el rechazo de peticiones y modificar la conversión automática de PEDIDO en HACER para que el agente distinguiese entre acciones propias y acciones pedidas.

Este proceso de control se ejecuta mientras que la agenda contenga algún acto. Esto es, el ciclo de control existe mientras tengo algo que hacer. Si por alguna razón particular fuese necesario implementar un agente con vida ilimitada, se podría implementar fácilmente. Bastaría con insertar en la agenda un acto de ejecución cuya condición tuviese una función *Preparado* que nunca se hiciese verdadera.

3.5 Comparación con otros modelos

ABC^2 se basa en una aproximación oportunista, modelo que se introdujo en el capítulo anterior con el de planificación tradicional y reactiva. En esta sección se van a resaltar las diferencias concretas de ABC^2 con el modelo tradicional de planificación y con el reactivo, y además con los ejemplos de sistemas oportunistas comentados en la Sección 2.1.3.

Los sistemas tradicionales se basan en el proceso de instanciación de los operadores para generar la solución, lo cual genera nuevos posibles estados. El proceso de búsqueda de la solución pasa por realizar un análisis de los estados futuribles para encontrar la secuencia adecuada de los mismos que lleve desde el estado inicial al final. Es decir, realizan un razonamiento hipotético sobre el estado.

En ABC^2 no existe el concepto de instanciación de operadores, ni se basa en la generación de estados hipotéticos, ni tampoco se realiza ningún razonamiento sobre estados futuribles. Las acciones se deciden en función de la situación real del mundo y de las oportunidades de actuación que se generan en cada momento. En ese sentido, es similar a los modelos reactivos. Sin embargo, los sistemas reactivos sólo generan una acción *óptima* a ejecutar en cada instante del tiempo. Por ejemplo, un robot dispara y ejecuta inmediatamente la acción "evitar obstáculo" si la distancia a un obstáculo es menor que un valor determinado. Es decir, los eventos producidos en tiempo de ejecución controlan las acciones del robot porque el modelo reactivo "cablea" la jerarquía de metas del agente.

En cambio, ABC^2 utiliza las condiciones de disparo sólo para permitir que las acciones puedan ejecutarse, lo que hace que en cada momento puedan identificarse distintas acciones. Así, el robot del ejemplo anterior controlado por ABC^2 puede no ejecutar "evitar obstáculo" en determinadas condiciones. Por ejemplo, si su meta en ese momento es detener al objeto que se está moviendo. Esto es, el robot puede o no ejecutar las acciones cuyas condiciones de disparo son ciertas dependiendo de sus

metas, que pueden modificarse mediante las heurísticas y no sólo por la definición de las habilidades. En resumen, un agente puede cambiar sus metas sin cambiar la forma en que sus acciones se disparan y además, los eventos en tiempo de ejecución activan, pero no controlan, las acciones del agente.

Una de las diferencias principales entre ABC^2 y los planificadores clásicos es la forma en que se especifican las metas. En los primeros se suelen definir las metas explícitamente, generalmente en forma de configuraciones del mundo, es decir, especificando una serie de características que se desea que se cumplan en el mundo. En ABC^2 se describen de forma implícita en la definición de las *Habilidades*. Lo cual implica que el par ($Habilidad_i$ - $meta_i$) es indivisible, al igual que lo es la $Habilidad_i$ de la lista $(Habilidad_i)^*$ de habilidades que la pueden hacer ejecutable. Cuando una habilidad no puede ejecutarse, todas las habilidades de su lista de necesidades se incluyen en la agenda. Esto quiere decir que se podrían considerar sus necesidades como precondiciones conjuntivas. Sin embargo, la inserción en la agenda no implica que todas sus precondiciones se tengan que ejecutar. Típicamente, sólo una de ellas es suficiente para hacer que su padre sea ejecutable. Así, la lista de necesidades de una habilidad se debería interpretar como un OR de posibilidades. Por contra, en los planificadores clásicos, cada operador del plan final obtenido tiene que ejecutarse obligatoriamente para que se obtengan las metas.

Otra diferencia es la forma en que se realiza la planificación. En los planificadores clásicos ésta se basa en la búsqueda en el árbol de posibles combinaciones de operadores que llevan desde la situación inicial a la final, donde el árbol se genera en función de los efectos que producen los operadores, eligiendo aquellos operadores que producen los efectos deseados y generando submetas para sus precondiciones. Todo ello gracias a que los efectos que los operadores producen en el mundo y las precondiciones necesarias para que se puedan ejecutar, se definen de forma declarativa y por tanto se puede razonar sobre ellos.

En ABC^2 esta búsqueda está "precompilada" en la *Lista de Necesidades* de cada habilidad. No es posible realizar ninguna búsqueda, porque el atributo *Ejecutar* de las habilidades no define explícitamente sus efectos. No hay creencias que añadir al estado del mundo, ni hechos que eliminar. Sólo produce una acción que puede tener un efecto no determinista. De la misma forma, el atributo *Preparado* representa el grado de certidumbre en el cumplimiento de sus precondiciones, que se pueden definir en función de cualquier información disponible para el agente.

Hay también algunas similitudes y diferencias en la forma en que se tratan las metas pendientes. Los planificadores clásicos expanden las precondiciones de los operadores como nuevas metas. En ABC^2 el contador *Llamadas* garantiza que cada meta se define sólo una vez. Además, se puede utilizar para saber qué habilidades son las más solicitadas en cada momento, de cara a su ejecución.

En cuanto a las implementaciones de otros sistemas, se podría considerar que el más similar es RAP [Firby, 1994], pues utiliza una agenda al igual que ABC^2. Este sistema se cataloga dentro de los sistemas de planificación jerárquica . Así, las tareas tienen una clausula *context* que podría considerarse similar a la condición *Preparado*, cada tarea tiene una lista de subtareas, etc. Sin embargo, el planteamiento es muy distinto. Las habilidades de ABC^2 no son directamente ejecutables, los actos lo son. Las habilidades representan acciones que el agente "sabe" ejecutar, no son acciones potenciales. Por eso, otro agente puede pedir al agente que ejecute un acto, puede informarle sobre los suyos, etc.

Los RAP de James Firby especifican un plan, una secuencia de acciones que el agente tiene que ejecutar para conseguir la meta establecida. Las habilidades de ABC^2 no tienen el concepto explícito de meta, ni tampoco implican la ejecución de otra serie de acciones. Son descripciones de cosas que el agente sabe hacer. Su lista de necesidades no implica que tengan que ejecutarse. Son enumeraciones de otras habilidades que podrían hacer que ésta se pudiese ejecutar.

Es decir, evaluar un acto de ABC^2 que conlleva la ejecución de una habilidad, no implica necesariamente que todas las acciones de la lista de necesidades se hayan ejecutado antes, ni que se vayan a ejecutar. El algoritmo de RAP sí hace estas suposiciones. De hecho usa la agenda únicamente para almacenar la lista de acciones que aún debe ejecutar, es decir, como lista de acciones pendientes. ABC^2 utiliza la agenda como lista de acciones potenciales, permitiendo que sea la heurística de selección de actos la que determine el flujo de control del agente.

La organización de la lista de necesidades, que puede verse como un árbol que parte de la meta global, puede recordar a los árboles TR propuestos por Nils J. Nilsson [Nilsson, 1994]. Sin embargo, estos árboles son completamente estáticos y están predefinidos. Esto es, el algoritmo de planificación se limita a elegir el nodo de menor profundidad que pueda activarse. En el modelo ABC^2 el árbol formado por la lista de necesidades no implica prioridad de ejecución. Las heurísticas de selección de la agenda permiten una mayor flexibilidad y por tanto pueden abordar tareas de mayor

complejidad.

Sin embargo, ABC^2 no puede utilizarse en entornos de tiempo real estricto, es decir, en aquellos en los que las restricciones temporales son críticas. En ese caso arquitecturas como CIRCA [Musliner *et al.*, 1994] que establecen duraciones máximas para las acciones son más adecuadas. Por otra parte, los problemas donde resultan útiles este tipo de consideraciones, en general el control de sistemas críticos, plantas de producción, etc. queda fuera del dominio para el que se ha diseñado este modelo. Tampoco es ABC^2 un modelo comparable con arquitecturas como TCA (*Task Control Architecture*), que están pensadas fundamentalmente para realizar las labores de *scheduling* [Simmons, 1992] de tareas.

Otra característica de ABC^2 es su generalidad. Por ejemplo, en la competición de la RoboCup muchas de las arquitecturas presentadas consistían básicamente en programas bien optimizados para dicho entorno pero difícilmente transportables a otros. ABC^2 parte de la idea contraria. Es un modelo que se puede aplicar a diversos dominios sin modificar su estructura fundamental.

En esta tesis se describe el uso de ABC^2 para el control de agentes software en el dominio de la RoboCup y para el control de mini-robots Khepera en tareas cooperativas, lo que lo distingue también de sistemas, como INTERRAP, que se han utilizado únicamente en un simulador "ad-hoc". Así como de muchas de las arquitecturas multiagente descritas, probadas sólo en entornos de simulación software, tipo MICE, con poco parecido con el mundo real.

También existe diferencia en su implementación. ABC^2 se ha implementado sin emplear ningún sistema de razonamiento de propósito general (no utiliza PRODIGY, ni OPS5, ni ningún otro sistema similar).

Capítulo 4

Utilizando el Modelo

Computer programming is an art form,
like the creation of poetry or music
Donald E. Knuth

Analizar y diseñar un sistema software para resolver un problema, o para diseñar un modelo, como es este caso, es una tarea que se suele realizar sin tener en cuenta las restricciones reales que impone la implementación física del mismo. Sin embargo, muchas veces la utilidad del sistema diseñado viene dada por la implementación del diseño realizado.

En el capítulo anterior se describieron una serie de algoritmos y de estructuras de datos resultado de analizar el problema del control de robots autónomos y de comparar los sistemas y aproximaciones más relevantes realizadas hasta la fecha. A ese conjunto de elementos se le denominó ABC^2. Al objeto de poder verificar la validez de dicho modelo, es necesario plasmarlo en forma de software que permita su evaluación en distintas situaciones. Este capítulo describe, a grandes rasgos, las características del software desarrollado para implementar el modelo ABC^2 y los condicionantes de su implementación, mostrando además un ejemplo del funcionamiento del sistema.

La estructura del capítulo es la siguiente: En la Sección 4.1 se describe la jerarquía de clases desarrollada en C++ que implementa el sistema, así como el resto de elementos principales del software desarrollado. La Sección 4.1.3 muestra los pasos necesarios para poder utilizar el software desarrollado y para poder ampliarlo. La Sección 4.2 explica, con un ejemplo, el funcionamiento de la implementación. Por último, en la Sección 4.3 se realizan algunas consideraciones sobre el ejemplo descrito.

4.1 Estructura del software desarrollado

La estructura del software desarrollado para implementar el modelo descrito en el capítulo anterior se ha diseñado con una serie de objetivos:

1. Generar una estructura básica que fuese utilizable en diversos tipos de problemas, para lo cual es muy importante separar las partes independientes del problema de aquellas específicas del mismo.

2. Garantizar que las habilidades, en principio específicas de cada tipo de problema, fuesen fácilmente adaptables y ampliables. Es decir, que la estructura básica del software considerase cada una de las habilidades como una entidad independiente.

3. Desarrollar un software fácil de mantener, para lo cual su estructura debería ser intuitiva y corresponder lo más fielmente posible con el modelo descrito.

4. Permitir que el estado del agente sea analizable. Esto es, que se pueda obtener una traza de las decisiones que el agente ha tomado y las condiciones en las que lo ha hecho.

4.1.1 Diseño de las clases

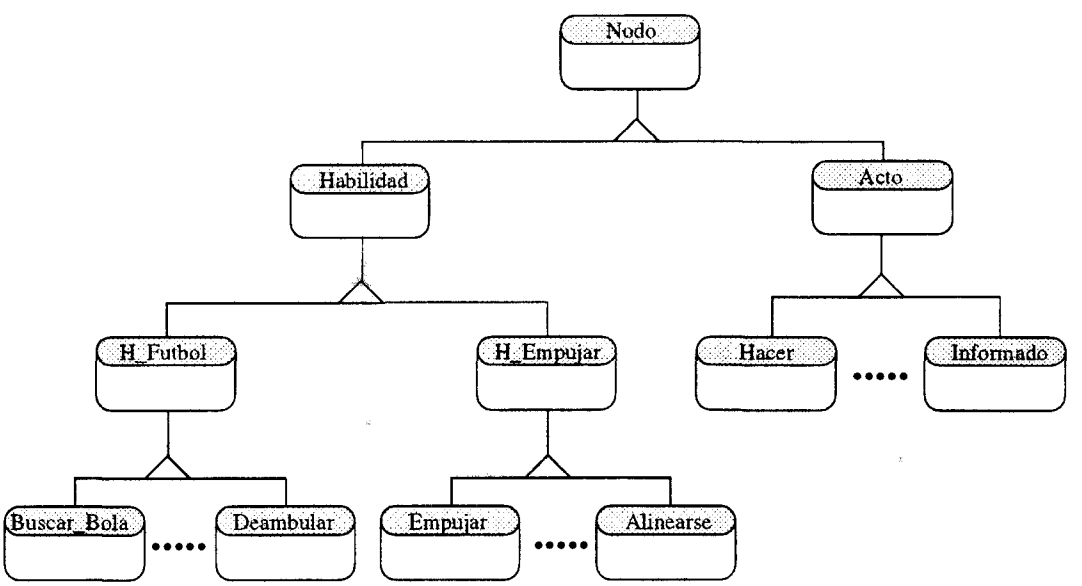

Figura 4.1: Jerarquía de herencia entre las clases implementadas.

Se decidió emplear la orientación a objetos como paradigma de programación. Siguiendo la metodología empleada (OMT [Rumbaugh *et al.*, 1991]), el paso fundamental

en la construcción del sistema es la identificación de las clases del mismo. La Figura 4.1 muestra la jerarquía básica de clases que se diseñó siguiendo el modelo descrito en el capítulo anterior. Esta jerarquía no incluye todas las clases creadas, pero sí es la jerarquía de herencia más importante, puesto que incluye la representación de los actos y las habilidades.

Ambas construcciones van a poder insertarse en listas, las primeras en la agenda, que en el fondo no es más que una lista con ciertos aditamentos como la extracción priorizada; las segundas en la lista de necesidades, que será una lista. Por ello, ambas clases se han definido como clases derivadas de la clase `Nodo` que implementa los atributos (punteros, por ejemplo) y métodos (acceso a los punteros y a la información) necesarios para poder ser insertadas en una lista.

Formalización 4.1 Definición de la clase *Habilidad*

```
class Habilidad: public nodo{
protected:
   lista* necesidades;
   float  peso;
   float  prioridad_inicial;
   char*  resultado;

public:
   skill(void);
   ~skill(void) {if (needs) delete (needs);};
   virtual float di_prioridad_inicial(void) {return (prioridad_inicial);}
   virtual float di_peso(void) {return (peso);}
   virtual void pon_peso_inicial(float p) {peso_inicial = p;}
   virtual void pon_peso(float p) {peso = p;}
   virtual lista* di_necesidades(void) {return (necesidades);}
   virtual char* di_nombre(void) {return("Ninguno\0");}
   virtual char* di_resultado(void) {return (resultado);}

   virtual void ejecutar(void);
   virtual int preparado(void);
};
```

En la Figura 4.1 se incluye únicamente el nombre de la clase. Los métodos y atributos se corresponden con los componentes descritos en la Sección 3.3. Como ejemplo, en la Formalización 4.1 se puede apreciar la definición completa de la clase `Habilidad`. Contiene los atributos descritos: `necesidades` declarado como una lista, que contendrá referencias a otras habilidades; la `prioridad_inicial`, que se le asigna

en el diseño del sistema; el método `preparado()` para describir la condición que permite la ejecución y el método `ejecutar()` que implementará la habilidad propiamente dicha.

La orientación a objetos permite de esta forma ocultar la complejidad de las operaciones. Por ejemplo, la Formalización 4.1 muestra la interfaz realizada para la clase `Habilidad`. Se puede apreciar que en su definición no aparece ninguna información relativa a los punteros para poder generar la lista de habilidades de un agente.

Sí aparecen, sin embargo, algunos otros elementos como `peso` que sirve para almacenar la *Prioridad* instantánea que calculan las reglas heurísticas. Conceptualmente, es probable que hubiese tenido más sentido como atributo de la clase `Acto`. Sin embargo, debido al tratamiento especial explicado de los actos de cooperación, se ha considerado más adecuado incluirlo en esta versión junto con el valor inicial.

La definición de la clase `Habilidad` descrita en la Formalización 4.1 está realizada en C++, que ha sido el lenguaje empleado para implementar la estructura básica de ABC^2. Como el tipo de plataforma donde se pretendía aplicar el modelo puede necesitar interaccionar con el hardware, como es el caso cuando se emplean robots, se decidió usar el lenguaje C++ en vez de cualquier otro orientado a objetos para facilitar el desarrollo del interfaz con el hardware y conseguir que fuese más eficiente.

Eso hace que se hayan empleado las particularidades de C++ para su construcción. Por ejemplo, la clase `Habilidad` tiene un atributo `resultado`, donde se almacenará el resultado de `ejecutar` la habilidad. La definición del mismo se ha hecho genérica, como una cadena de caracteres. Por último, todos los atributos se declaran como `protected` para que sean conocidos por sus clases derivadas, pero para que tengan que ser accedidos de forma controlada desde otras clases (mediante los métodos `pon_*` para modificar y `di_*` para consultar).

Por otro lado, tanto la clase `Habilidad`, como la clase `Acto` tienen a su vez clases que heredan de ellas. Por ejemplo, de la primera se ha especializado una clase, denominada abreviadamente `H_Futbol`, que servirá para incluir en ella los métodos y atributos específicos del dominio del fútbol entre robots. Esta clase se describe con más detalle en el Capítulo 6. Especializaciones de esta clase serán las acciones básicas de ese dominio, como el comportamiento `Buscar_Bola` descrito en la Formalización 6.3. Por su parte, la clase `H_Empujar`, también especialización de `Habilidad`, representa la base de todas las habilidades necesarias para implementar las acciones básicas del dominio de cooperación que se describe en el siguiente capítulo.

Por su parte, de la clase `Acto` derivan las clases que implementan los distintos tipos

de acto que se describieron en el capítulo anterior: `HACER`, `PEDIR`, etc. Cada uno de ellos con sus componentes especializados y todos ellos insertables en la `Agenda`, puesto que son derivados de `Nodo`.

4.1.2 Relación entre los elementos

Una vez decidido el paradigma a emplear, realizado el diseño de las clases, y definida la estructura básica de las clases, resta fijar la relación entre ellas. La Figura 4.2 muestra dicha relación. Así, la clase `Agente` representaría una entidad (robot o en general agente) controlada mediante ABC^2. Dicho agente contiene, según el modelo, una agenda (objeto clase `Agenda`), que su vez puede contener una serie de actos (objetos de la clase `acto`) cuyo número puede ir de 0 (cuando está vacía y termina la existencia del agente) a n (en plena ejecución). La relación *0:n* se ha representado con el círculo hueco.

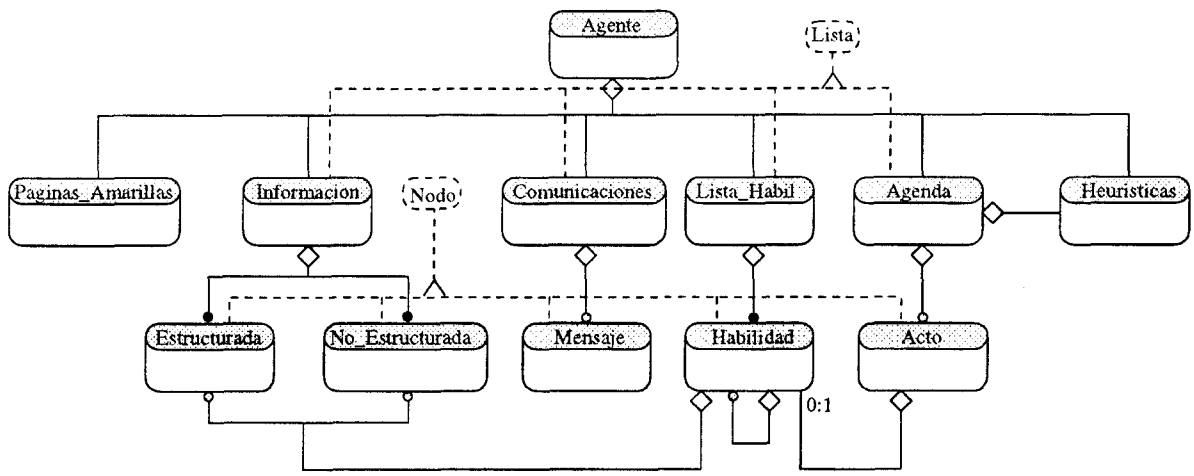

Figura 4.2: Relaciones entre las clases.

Además, el agente tendrá una lista de habilidades, que se representa mediante la clase `Lista_Habil`, que contendrá al menos una habilidad, la inicial. Para representar esta relación *1:n* se ha utilizado el círculo relleno. Se ha incluido de nuevo en la Figura 4.2, con líneas discontinuas, la relación de herencia entre la clase `Lista` y varios de los componentes del agente. Igualmente se ha representado la herencia de la clase `Nodo`, para explicitar las ventajas de la herencia a la hora de diseñar software relativamente complejo.

Las habilidades, por su parte, tienen una lista de necesidades, lo que hace que se represente una relación reflexiva en su clase. Y pueden utilizar, o no, las informaciones

tanto para implementar su método `preparado` como para hacer cálculos en `ejecutar`.

Otro de los componentes fundamentales del agente es la información que conoce del entorno. En el esquema de la Figura 4.2 se muestra la clase `Informacion`, que estará compuesta por una serie de informaciones estructuradas y no estructuradas. La implementación de esas informaciones dependerá del dominio en el que se empleen.

Formalización 4.2 Ejemplo de información estructurada

```
class PosicionRelativa {
 protected:
  float distancia, distChng;
  float angulo, dirChng;
  int tiempo;
 public:
  PosicionRelativa();
  void Olvidar()
  virtual void PonDistancia(float dist)      {distancia=dist;}
  virtual void PonAngulo(float ang)          {angulo=ang;}
  virtual void PonDistAngTiempo(float, float, int );
  virtual void PonTodo(float, float, float, float, int);
  virtual void ImprimirPosicion(FILE *);
  float DiDistancia()                        {return distancia;}
  float DiAngulo()                           {return angulo;}
  float DiDistChng()                         {return distChng;}
  float DiDirChng()                          {return dirChng;}
  int DiTiempo()                             {return tiempo;}
};
```

Así, la Formalización 4.2 muestra un ejemplo de información estructurada para almacenar la posición relativa, respecto del agente, de un objeto en el dominio de la RoboCup que se analizará en el Capítulo 6.

Las heurísticas que rigen el proceso de selección de la agenda son otro de los componentes fundamentales. En principio, el modelo no presupone que estén implementadas de ninguna forma en particular. De hecho se ha representado como una clase relacionada con la agenda para indicar que simplemente deben asignar una prioridad a los actos contenidos en ella, el mecanismo con el que lo haga es independiente del resto del sistema. En concreto, en los dos próximos capítulos se describen como reglas evaluadas mediante lógica borrosa.

El agente dispondrá, además, del resto de las componentes descritas en el mode-

lo, como son: las páginas amarillas, con información sobre sus compañeros, o la clase `Comunicaciones`, que tendrá distintas implementaciones según el sistema de comunicaciones disponible en cada entorno. Esta clase gestiona además los `Mensajes` entre los robots, pudiendo guardar una lista de ellos, o gestionarla según se necesite.

En la definición de los actos realizada al formalizar el modelo se indicó que algunos de ellos referencian a una habilidad. Por ejemplo, un acto de tipo `HACER` especifica qué habilidad va a ejecutar. Un acto `INFORMAR` no tiene por qué. Dicha relación está también reflejada en la figura. A su vez, las habilidades pueden hacer uso de las informaciones contenidas en `Informacion`.

Las relaciones definidas en la Figura 4.2 se refieren a relaciones de diseño. El uso de C++ como lenguaje de programación, como cualquier otro lenguaje, aporta sus particularidades a la hora de implementar el software. Por ejemplo, la Figura 4.3 muestra una simplificación de las relaciones entre los distintos objetos en memoria.

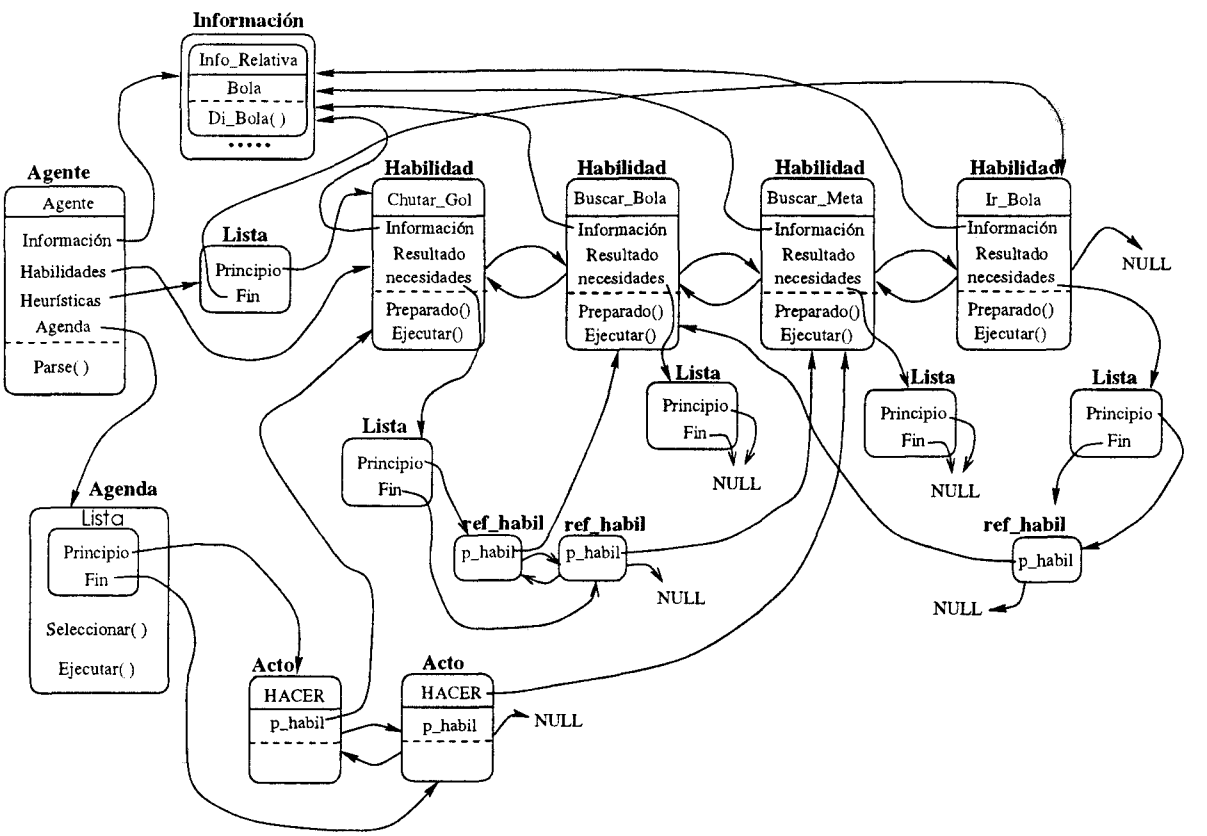

Figura 4.3: Relaciones dinámicas entre los objetos.

Por ejemplo, en la figura aparece un objeto, denominado `ref_habil`, que no se ha descrito anteriormente. La utilidad de ese objeto es evitar tener copias repetidas

de la misma información en memoria. Es decir, en memoria sólo existe una copia de cada habilidad. Si un acto "referencia" una habilidad, lo hace a través de un objeto `ref_habil`; de la misma forma, la lista de necesidades de una habilidad es una lista de referencias, etc. Todo ello con la idea de generar el código más compacto, modular y eficiente posible.

El software construido ha resultado ser bastante reducido, sobre todo porque no se ha implementado ningún tipo de interfaz gráfico de usuario. El código está formado por unas 5000 líneas de código C++, descontando comentarios; más unas 2000 líneas en C que implementan el sistema de razonamiento borroso. Esta última librería permite utilizar un sistema básico de razonamiento borroso (que se explica en el Apéndice A) de forma flexible y es el resultado de un trabajo conjunto con José Manuel Molina [Molina, 1997].

Por último, indicar que en las figuras y comentarios precedentes no se han incluido, por cuestiones de espacio y relevancia, todas las clases implementadas. Por ejemplo, para el experimento descrito en el Capítulo 5, se implementó una clase `sensores` que modela el comportamiento de los sensores. Modificando levemente esa clase, se pudo emplear el resto del código, sin modificaciones, al pasar del simulador al robot real. Lo cual da una idea de la modularidad del sistema y del buen grado de encapsulamiento de la información.

4.1.3 Configuración

Una de las restricciones de diseño de ABC^2 es que pueda ser un sistema "empotrable", es decir, capaz de ejecutar en un hardware específico, por ejemplo en un robot. Eso hace que no sea una aplicación convencional, con una interfaz de usuario para las entradas, unos cálculos y otra interfaz para los resultados.

Sin embargo, también se deseaba que el software desarrollado fuese configurable, tanto para poder indicar a los agentes qué tareas desarrollar como para configurar las habilidades de que se dispone, las de sus compañeros, etc. Por ello se ha intentado que el sistema sea lo más parametrizable posible y que todas las estructuras de datos sean dinámicas, de forma que no haya un número máximo de habilidades, ni de actos en la agenda, ni de elementos en la lista de necesidades, etc.

La solución elegida en la implementación ha sido que los agentes se configuren mediante ficheros de texto para hacer más intuitiva su comprensión. En el Apéndice B se puede encontrar la definición exhaustiva de varios agentes para un dominio de robots

que juegan al fútbol.

Además, como todo sistema experimental, la implementación debe facilitar el análisis de su funcionamiento. Así, se ha previsto la posibilidad de generar ficheros de traza del comportamiento de los agentes. Estos ficheros contendrán la información relativa a las decisiones de cada uno de los jugadores.

La traza a su vez es configurable, puediendo indicarse qué información se quiere almacenar. Ello es debido a que en ocasiones puede ser demasiada la información generada. Por ejemplo, durante la celebración de un partido de la competición de fútbol entre robots, la generación del fichero de traza sobrepasaba los 4 megabytes de información por jugador.

El formato de la traza es, por ejemplo, el mostrado en la Formalización 4.3. Esta fracción corresponde al fichero de traza generado por el agente que tiene que efectuar el saque desde el centro del campo en el dominio del fútbol. Corresponde con los instantes iniciales, como se puede apreciar por la cuarta línea, donde aparece el mensaje enviado por el árbitro, para indicar que ha comenzado el partido.

Formalización 4.3 Traza del comportamiento de un jugador

```
(see 0 ((goal r) 62.8 0) ((flag r t) 70.8 -28) ((flag r b) 70.8 28)
((flag p r t) 50.4 -23) ((flag p r c) 46.1 0) ((flag p r b) 50.4 23)
((ball) 10 0 0 0) ((line r) 62.8 -89))
(hear 0 referee kick_off_l)
(change_view narrow high)

(see 1 ((goal r) 62.8 0) ((flag p r c) 46.1 0) ((ball) 10 0 0 0)
((line r) 62.8 -89))
Contenido de la Agenda:
  1) HACER: Habil: Ganar_Partido Peso: 0.900  Preparado 0
  2) HACER: Habil: Chutar_Gol Peso: 0.800  Preparado 0
  3) HACER: Habil: Ir_Posicion Peso: -0.300 Preparado 1
  4) HACER: Habil: Ir_Bola Peso: 1.700 Preparado 1
Eval Ir_Bola, Resultado: (dash 100.0)

(see 3 ((goal r) 60.9 0) ((flag p r c) 44.7 0) ((ball) 9 0 -0.18 0)
((line r) 60.9 -89))
```

Esta formalización muestra el estado resumido del agente indicandose el contenido de la agenda, , donde se ordenan los actos por antigüedad, en cada instante. Para cada acto se indica si se puede ejecutar o no (si es aplicable).

Para los actos de tipo HACER se muestra el nombre de la habilidad a ejecutar y

el peso. En los no aplicables corresponde con la prioridad inicial (mostrada en la Formalización B.3). En los aplicables, el peso es el resultado de aplicar las heurísticas en ese instante.

Por tanto, la traza mostrada indicaría que en la situación en la que se encuentra ese agente, definida por la información disponible (`(see 0 ((goal r) ...)`. El sistema tiene dos opciones: `Ir_Posicion`, cuya prioridad inicial según la Formalización B.3 del Apéndice B, era 1, e `Ir_Bola` que inicialmente tenía una prioridad 0.7. Con la aplicación de las heurísticas, sin embargo, el más interesante pasa a ser `Ir_Bola`, que es el que se ejecuta, enviándose el comando `(dash 100.0)` que hace avanzar al robot con la máxima velocidad posible.

El resultado de ejecutar esa habilidad se refleja en la información visual recibida posteriormente `(see 3 ... ((ball) 9 ...`, donde se puede apreciar cómo la distancia a la bola se ha reducido de 10 a 9. La siguiente sección especifica con más detalle todos estos pasos.

4.2 Ejemplo de funcionamiento

Para explicar el funcionamiento del modelo se ha utilizado un experimento, descrito en detalle en el Capítulo 5, en el que dos robots tienen que empujar de forma coordinada una caja. Para simplificar el problema se supone que los puntos donde los robots tienen que empujar son fijos.

En primer lugar, hubo que definir las habilidades necesarias. Para ello se implementaron: `Empujar`, que consiste básicamente en avanzar (estando pegado al objeto), `Alinearse` que es capaz de colocarse perpendicularmente al punto donde debe empujar, `Buscar_Caja` que es capaz de encontrar su punto de alineación y `Evitar_Obstaculo`, que le permite moverse de forma aleatoria por el mundo sin chocar con los obstáculos. La organización de las listas de necesidades se muestra en forma de árbol (las necesidades de cada habilidad se han encadenado con las necesidades de éstas y así sucesivamente) en la parte superior izquierda de la Figura 4.4.

Una vez definidas las habilidades hay que diseñar las heurísticas. Supóngase que se simplifican a una simple regla que diga: "Seleccionar de la agenda aquellos actos cuyo valor de *Prioridad* sea el más alto". Es decir, no se produce ninguna variación al valor inicial del campo *Prioridad*

Supóngase además que la *Información* de que dispone el robot es el conjunto de

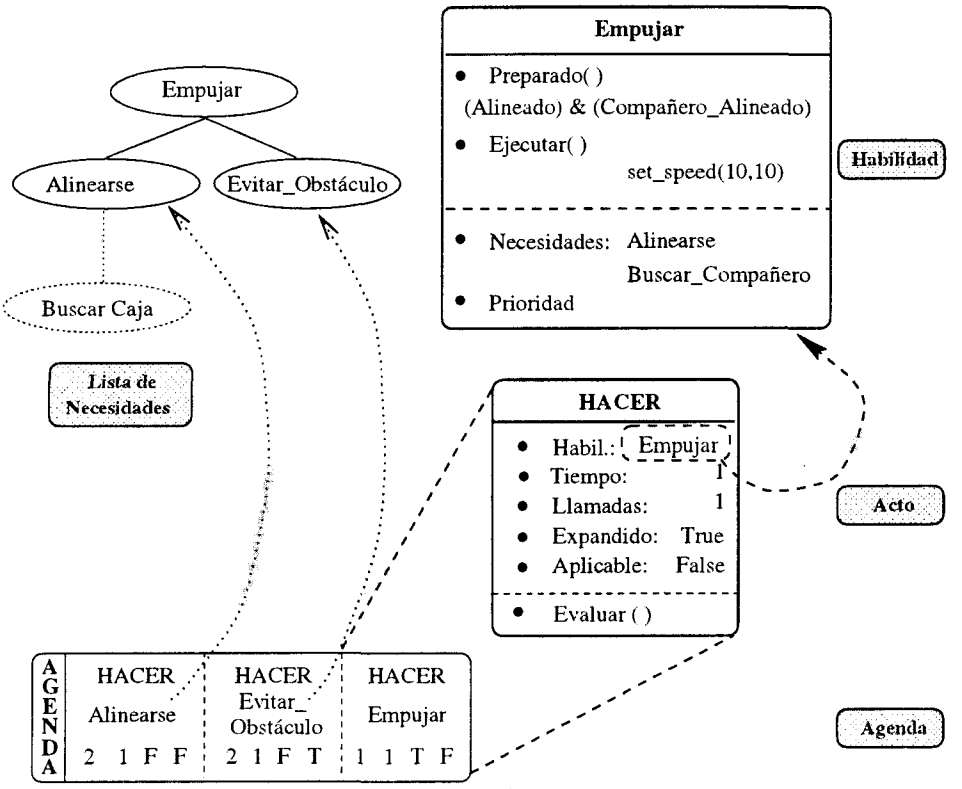

Figura 4.4: Relaciones entre las habilidades, los actos y la agenda.

valores recibidos de los sensores y la información de si su compañero está alineado o no. Supóngase que esta última información se genera mediante un intercambio periódico de mensajes generado como actos `INFORMAR - INFORMADO`.

Para empezar, los dos robots deben inicializarse, es decir, empezar a existir. Para ello, dependiendo del tipo de problema se les podría dar cierta información; por ejemplo su posición en el mundo, el número de compañeros, sus posiciones iniciales etc. En este ejemplo sencillo no son necesarias todas estas informaciones, por lo que la inicialización consistirá simplemente en poner a FALSE el valor del atributo de información `Compañero_Alineado`. Lo que sí es necesario es insertar la tarea inicial. Esto se realiza mediante la inserción del acto [`HACER:Empujar`] en la agenda, con el contador de *Tiempo* a uno, el indicador *Expandido* a FALSE y el atributo *Llamado* a uno porque es el programador el que solicita al robot que realice la acción. El valor de *Preparado* no hace falta inicializarlo, pues se calcula cada vez que se desee consultar.

La forma en que funciona el ciclo de control es la siguiente. Primero se seleccionan los actos aplicables; en el caso del acto [`HACER:Empujar`] se hace consultando el atributo *Preparado* de la habilidad `Empujar`. Como refleja la figura, una posible implementación de dicha condición es comprobar si el compañero está alineado y si él mismo lo está. Si

un acto resulta no ser aplicable, como es el caso, se comprueba su indicador *Expandido*. Si no se había expandido, las habilidades contenidas en la lista de necesidades de la habilidad contenida en él se insertan en la agenda como [HACER:<habilidad>]. En este caso, [HACER:Alinearse] y [HACER:Evitar_Obstáculo]. Cada uno de ellos inicializado adecuadamente, con *Llamado* a uno, *Tiempo* a dos y *Expandido* a FALSE.

La situación en este instante es la que aparece reflejada en la Figura 4.4 que muestra el árbol de necesidades en la parte superior izquierda. Se ha marcado de forma distinta la habilidad Buscar_Caja para indicar que no está en la Agenda. En la parte inferior se muestra el contenido de la agenda en ese instante, indicando el tipo de acto, el nombre de las habilidades contenidas en ellos y los valores de los atributos del acto.

Dentro de la Figura 4.4, en el centro-derecha, se muestra en más detalle (con los nombres de los atributos) uno de los actos contenidos en la agenda, concretamente [HACER:Empujar]. La parte superior derecha muestra en detalle los campos de la habilidad Empujar. Igualmente, los actos [HACER:Alinearse] y [HACER:Evitar_Obstaculo] contendrán las correspondientes habilidades.

El proceso de incorporación de las necesidades de una habilidad en forma de actos [HACER:<habilidad>] comprueba que ese acto no hubiese sido añadido antes y siguiese en la agenda. Si ya estaba en la agenda, el contador *Llamadas* de ese acto se incrementa y no se inserta de nuevo. En otro caso, se introduce un nuevo acto en la agenda.

Por el contrario, si un acto resulta ejecutable, las heurísticas lo seleccionan como el más apropiado para su ejecución y si había sido previamente expandido, entonces se comprueba cuántas de sus necesidades permanecen en la agenda y a ésas se les decrementa el contador *Llamadas*. Es decir, se indica que ya no necesita que sus necesidades se ejecuten porque es ejecutable.

Al mismo tiempo que se comprueban cuáles son los actos aplicables, se eliminan de la agenda aquellos actos cuyo contador de *Llamadas* sea cero, esto es, aquellos actos que nadie necesita. Por eso el acto inicial se inserta con dicho contador a uno.

Una vez que se ha depurado la agenda y se han elegido los actos aplicables, las heurísticas se pueden aplicar. El resultado de evaluar las heurísticas será el acto a ejecutar. Para ello lo que hacen es calcular un "peso" para cada uno de los actos y elegir aquél que tenga el mayor.

Según los pasos analizados, la forma en que se insertan los actos en la agenda es, bien como meta inicial, o bien como necesidades de otro acto. También se pueden insertar actos como resultado de la evaluación de otros actos o como resultado de la ejecución

de la habilidad contenida en ellos. Por ejemplo, la ejecución de la habilidad `Alinearse` de la Figura 4.4, genera la inserción de un acto `[PEDIR: RobotB, Alinearse]`, en donde la información sobre las habilidades del otro robot se obtienen de las *Páginas Amarillas.*

La evaluación de este acto de cooperación resulta en el envío de un mensaje al otro robot. Lo cual, por su parte, generaría un acto `[PEDIDO: Alinearse]` en el `RobotB`, que se trataría como se ha indicado anteriormente.

Este es el otro mecanismo por el que se pueden insertar actos en la agenda: por peticiones de otros agentes. Además, es la forma en que se integran las acciones propias con las pedidas por otros agentes: la inserción de actos en la agenda por los agentes con los que colabora. La gestión de esa colaboración se encomienda a las heurísticas.

En la implementación realizada, el tratamiento por las heurísticas de los actos de cooperación se ha simplificado de forma que sólo se contempla la posibilidad de aceptar la petición, si el robot está en la lista de quienes se aceptan peticiones, o desecharla si no lo está. Dicha lista, por simplicidad, se ha hecho coincidir con la lista de robots de los que se conocen habilidades, es decir, las *Páginas Amarillas.*

En este caso, significaría que el `RobotB` aceptaría la petición y la evaluación del acto `[PEDIDO:Alinearse]` resultaría en generación de un `[HACER:Alinearse]` que se trataría según los algoritmos anteriores.

De igual forma, los actos de información se aceptan y envían sólo a los robots que estén incluidos en las *Páginas Amarillas.* Su evaluación dará como resultado la modificación de las informaciones contenidas en la estructura *Información* del agente que recibe un acto de tipo `INFORMADO`.

Por ejemplo, el `RobotA`, además de la petición anterior, enviaría un mensaje del tipo `[INFORMAR: <RobotB, Alineado = True>]`. Su aceptación resultaría en un acto `[INFORMADO: <RobotA, Alineado = True>]` cuya evaluación resultaría en la asignación del valor `True` a la información `Compañero_Alineado`, que se usa en la Figura 4.4 para definir la condición *Preparado* del acto `Empujar`.

4.3 Consideraciones

Según el ejemplo anterior, se podría afirmar que el modelo ABC^2 asume, en parte, el principio de los procesos de Markov, es decir, la decisión tomada en cada instante es independiente de las anteriores. Así, en el proceso de evaluación de las heurísticas,

éstas seleccionan de entre los actos preparados aquél que se ejecutará teniendo en cuenta la información de ese instante y sin realizar ningún análisis explícito de la secuencia anterior de acciones.

Sin embargo, no se sigue este principio en el mecanismo de inserción de nuevas acciones en la agenda, donde se analiza si la acción ya estaba en la agenda, en cuyo caso se incrementa el contador *Llamadas*. Este contador representa por tanto parte de la historia de esa acción, en concreto cuántas otras acciones la necesitan.

Además, los actos almacenan también parte de su propia historia al guardar la información relativa al momento de su inserción. Dicha información sí es tenida en cuenta por las heurísticas (por eso se dice que asume Markov sólo en parte) a la hora de seleccionar el acto a ejecutar. El modelo también permite realizar un análisis no markoviano, simplemente insertando información en la memoria y recuperándola más tarde, tomando decisiones respecto a esa información. Es decir, el modelo en principio permite que sea el diseñador de la aplicación concreta quién decida.

Otro tipo de consideraciones que se pueden hacer con respecto al modelo se refieren al costo de los actos. ABC^2 no impone ninguna restricción al tipo de habilidades que se definen, ni a los mecanismos que las implementan. Puede tratarse de una simple asignación de valores predeterminados a los actuadores, el resultado de la evaluación de un sistema de reglas borrosas, etc.

Esta libertad puede suponer, sin embargo, problemas en ciertas circunstancias. El modelo no considera el coste computacional de los cálculos necesarios para implementar las habilidades incluidas en los actos `HACER`. Tampoco considera el coste de las comunicaciones en los actos `PEDIR` o `INFORMAR`. Dicho de otra forma, el modelo ABC^2 supone que el coste de las acciones es nulo o, más en general, equivalente (uniforme) en todas ellas.

En los experimentos realizados, que se describen en los siguientes capítulos, esta suposición ha demostrado ser aceptable. Sin embargo, la implementación de habilidades de costo relativo muy elevado respecto de las demás (por ejemplo, que utilicen un proceso tradicional de búsqueda) podría hacer necesario añadir el costo estimado a las habilidades y hacer que este costo sea utilizado por las heurísticas para elegir el acto a evaluar.

Una de las manifestaciones más típicas del costo de las habilidades es la duración de las mismas. Según las consideraciones anteriores se ha supuesto que la duración es uniforme para todas ellas, por lo tanto el funcionamiento temporal de un sistema

basado en ABC^2 no depende de la ejecución de las mismas. Cabe plantearse entonces si el funcionamiento es síncrono o asíncrono con respecto a la lectura de los sensores.

En principio, como mostraba la Figura 3.1, el modelo sigue un esquema tradicional en el que se procesa la información recibida de los sensores, se toma la decisión sobre que acción realizar (qué acto evaluar) y se envía el resultado de la misma a los sensores. Sin embargo, el modelo no presupone ningún tipo de sincronía de funcionamiento del sistema con la lectura de los sensores. La única suposición en ese sentido que se hace en el modelo es que en cada ciclo se toma una acción y sólo una. En ningún momento se establece que el ciclo de control se dispare por la lectura de los sensores. Es más, allí donde sea posible, puede resultar conveniente separar el proceso de interpretación de la información de los sensores, del proceso de toma de decisiones que simplemente utiliza la información ya procesada.

En cualquier caso, la mejor forma de evaluar la significación de estas suposiciones es mediante la prueba del modelo en diversos entornos. Los siguientes capítulos realizan esa tarea. Así se describen dos dominios donde se ha aplicado el modelo ABC^2. El primero, que se analiza en el siguiente capítulo, corresponde a una versión del problema clásico de cooperación entre dos robots para el traslado coordinado de un objeto. El segundo, que se analiza en el Capítulo 6, permite evaluar el rendimiento del modelo en un dominio competitivo (robo-fútbol) y que involucra a mayor número de agentes (once jugadores).

Capítulo 5

Experimentación en un Entorno de Cooperación

"Contrariwise," continued Tweedledee, "if it was so, it might be, and if it were so, it would be; but as it isn't, it ain't. That's logic!"

Lewis Carroll

El mecanismo básico para la validación de teorías según el método científico ha sido durante los últimos siglos la experimentación. En las ciencias básicas, los procedimientos de experimentación están firmemente establecidos y reconocidos; sin embargo, en las nuevas ciencias, como la informática, no lo están tanto. Encontrar los mecanismos que permitan afirmar irrefutablemente que los objetivos perseguidos con el modelo presentado se cumplen, no es fácil. El camino elegido ha sido probar implementaciones del modelo en entornos diversos y comprobar cuáles son las ventajas de su aplicación y cuáles los defectos.

Este capítulo presenta la aplicación de ABC^2 al problema clásico de cooperación entre dos robots para el traslado cooperativo de un objeto. Para ello, en la primera sección se describe el experimento y se presentan las dos versiones del mismo que se han realizado: una simulada y otra con robots reales. La segunda sección describe las habilidades empleadas y la forma en que han sido creadas. Así, se describen habilidades implementadas de dos formas distintas: como controladores borrosos diseñados por el autor y mediante aprendizaje automático. En la última sección se discuten los resultados del experimento realizado.

5.1 Definición del experimento

Elegir el tipo de pruebas que se van a realizar para evaluar un modelo como ABC^2 tiene una importancia particular. Por ejemplo, una de las críticas más habituales a las aproximaciones simbólicas ha sido precisamente que sus experimentos se han realizado siempre en entornos bien estructurados. Por otra parte, los experimentos con robots reales tienen múltiples problemas, desde presupuestarios hasta de comparación con otros modelos. Por ello la definición de los problemas debe ser cuidadosa, así como los objetivos que se persiguen con cada uno.

5.1.1 Diseño del Experimento

A la hora de diseñar experimentos de cualquier tipo se suele comenzar por el supuesto más sencillo. En el caso de las tareas cooperativas entre robots, el más simple es la colaboración entre ellos. Dentro de las tareas que dos robots como los disponibles pueden realizar conjuntamente, la de empujar un objeto es a su vez la más asequible. Además éste es uno de los problemas típicos para probar este tipo de modelos.

Este problema ha sido empleado como campo de prueba tanto por la escuela simbólica, por ejemplo [Noreils, 1993], donde se describe cómo dos robots se coordinan para empujar un objeto; como por la reactiva, por ejemplo en [Kube and Zhang, 1990], donde se describen los comportamientos reactivos realizados para que un grupo de robots trasladen un objeto identificado por una luz.

La diferencia entre ambos tipos de planteamiento es que en el primer caso existe el concepto explícito de objeto, de traslado, etc. mientras que en el segundo la acción coordinada surge de los comportamientos individuales de buscar la luz y tratar de acercarse a ella, lo que hace que los robots empujen el objeto.

La versión propuesta de este tipo de entorno tiene como objetivo principal demostrar la validez del modelo ABC^2. Es decir, verificar que el modelo descrito en los dos capítulos anteriores realmente es capaz de controlar un robot autónomo y que además le permite colaborar con otro.

A la hora de evaluar el rendimiento de un sistema se pueden emplear métodos cuantitativos o cualitativos. En un entorno como el propuesto la evaluación cuantitativa parece poco significativa. Medir el tiempo empleado en trasladar el objeto, el tiempo en encontrarlo, etc. depende más del diseño de las habilidades (velocidades, compromiso con la seguridad de los robots, etc.) que de la idoneidad del modelo. Por ello los

experimentos se han centrado en la evaluación cualitativa del modelo.

Se fijó como objetivo del experimento verificar algunas de las cualidades del modelo. En primer lugar comprobar que se puede emplear tanto para controlar robots reales como simulados, es decir, que se trata de un modelo general. Comprobar que es capaz de integrar habilidades generadas de distinta forma, es decir, que es un modelo multi-paradigma. Así, en el experimento se utilizaron controladores borrosos diseñados heurísticamente y mediante aprendizaje genético. Además, este experimento permite verificar el funcionamiento de los distintos tipos de actos, así como los dos tipos de representación de la información.

Por último, el experimento permite evaluar el comportamiento del modelo desde el punto de vista de su fiabilidad. Es decir, comprobar que los agentes son capaces de realizar la tarea encomendada ante distintas configuraciones del entorno (distribución de obstáculos, puntos de partida, etc.).

La aproximación seguida para el desarrollo del experimento ha sido la propuesta en [Arkin, 1993], es decir, en primer lugar se diseña el experimento en un simulador y a continuación se verifica con los robots reales. Este procedimiento equivaldría al método científico de formular las hipótesis, "Si en el simulador funciona ..." y verificarlas experimentalmente "...en la realidad también".

Para ello, en primer lugar se realizaron los experimentos en un simulador diseñado a tal efecto por el autor y por sus colegas del Laboratorio de Agentes Inteligentes [Sommaruga *et al.*, 1996]. Dicho simulador, denominado SimDAI, se describe en la siguiente sección.

5.1.2 Simulador SimDAI

SimDAI, acrónimo de *Simulator for Distributed Artificial Intelligence*, (Simulador para Inteligencia Artificial), es un paquete de software que permite la simulación de varios robots móviles en un entorno de dos dimensiones donde pueden existir diferentes objetos.

Este sistema proporciona soporte para tres tipos de actividades: la simulación de los robots (procesos robot en la Figura 5.1), la simulación de una determinada configuración del entorno donde se mueven (proceso mundo) y la monitorización de dicho entorno (procesos monitor en la Figura 5.1).

Cada una de estas actividades se desarrolla por medio de procesos distintos, como muestra la Figura 5.1. Además, SimDAI es un sistema distribuido, por lo que cada uno

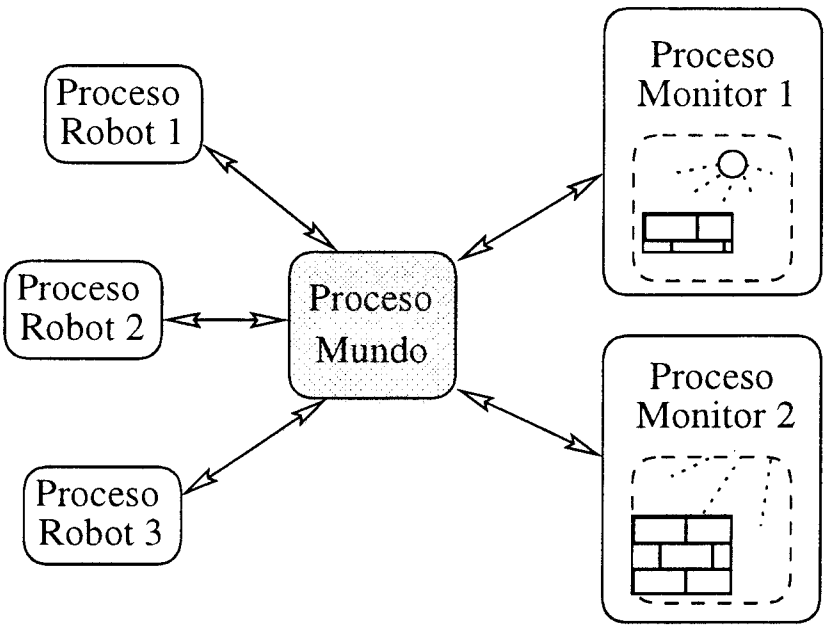

Figura 5.1: Estructura del simulador SimDAI.

de esos procesos puede ejecutarse en máquinas distintas y conectarse con los demás a través de la red, por medio de *sockets* BSD.

La estructura del sistema se centra en el proceso que simula el *mundo*, y que al igual que éste será único. Este proceso se encarga de gestionar la representación del mundo (actualizar las posiciones de los robots, detectar choques, etc.). La definición del mundo la realiza el diseñador del experimento, pudiendo fijar el número y características de los objetos que aparecerán. Estas configuraciones pueden almacenarse y reutilizarse, de forma que se puedan repetir los experimentos bajo las mismas condiciones.

Para visualizar el estado del mundo en un determinado momento se emplean procesos del tipo *monitor*. Puede existir cualquier número de ellos, cada uno visualizando o bien una porción del mundo, o bien la misma porción a distinta escala, etc. (como muestran los monitores 1 y 2 de la Figura 5.1).

De la simulación de los agentes propiamente dichos se encargan los procesos *robot*, de los cuales también puede existir cualquier número en el mundo. En principio, los robots simulados son del tipo "tortuga" [Mckerrow, 1991], o lo que es lo mismo, dotados de dos ruedas independientes (para girar hay que hacer que una vaya más rápida que la otra) y un conjunto de sensores definible por el usuario en cuanto a tipo, colocación, orientación y número.

En principio, se han definido tres tipos de sensores denominados: sonar, que barre sectores del mundo informando de su grado de ocupación; láser, que hace lo propio

pero en una línea; y GPS, que informa al robot de su posición en el mundo. El número de ellos es variable pudiéndose configurar cada uno de ellos de forma individual.

Cada uno de los robots es completamente independiente, con sus propias dimensiones, color, nombre, etc. Se construyen independientemente del mundo y sólo pueden comunicarse con él a través de los mensajes pre-establecidos que representan la lectura de los sensores, las velocidades para sus ruedas y los mensajes que deseen enviar a otros robots. Esto hace que los robots puedan activarse en cualquier orden, y que puedan conectarse y desconectarse dinámicamente.

Los procesos robot pueden comunicarse entre ellos utilizando una serie de primitivas prefijadas para enviar mensajes. Todos los mensajes se envían al proceso mundo y éste se encarga de enviarlos al destinatario correspondiente. Se eligió este mecanismo de comunicación porque permite añadir condiciones como el ruido, la pérdida de mensajes, etc.

Para realizar los experimentos en el simulador SimDAI se ha diseñado un mundo simulado, como muestra la Figura 5.2. El objetivo era imitar el entorno en el que se utilizaron los robot reales y que se muestra en la Figura 5.8.

El aspecto del entorno simulado se puede apreciar en la ventana superior derecha de la Figura 5.2. Dicha ventana corresponde con un proceso monitor. En los experimentos realizados se ha utilizado un objeto rectangular, con dos triángulos más claros en su interior. Esos serán los puntos donde los robots deben alinearse para empujar.

La ventana superior izquierda de la Figura 5.2 corresponde con la interfaz de un proceso robot. Muestra la información de las lecturas de los sensores y permite modificar algunas de las características de la representación del robot en el proceso monitor, como si el robot deja el rastro de su trayectoria o no, el barrido de sus sensores, etc. Dichas modificaciones tendrán una correspondencia directa e inmediata en los procesos monitor. Por ejemplo, la ventana del proceso monitor muestra un robot con el rastro y el barrido de sensores activado.

El proceso mundo, por su parte, tiene también su propia interfaz. En la ventana correspondiente (inferior derecha) se muestran informaciones como el número y nombre de los robots incluidos en él, la frecuencia de la simulación, etc. Además tiene también varios menús para manejar las diversas opciones.

Para terminar la descripción del simulador, indicar que se diseñó con la idea de simular los robots Khepera. Los mismos controladores borrosos que se implementaron para controlar los robots simulados se han empleado luego en los robots reales, lo que

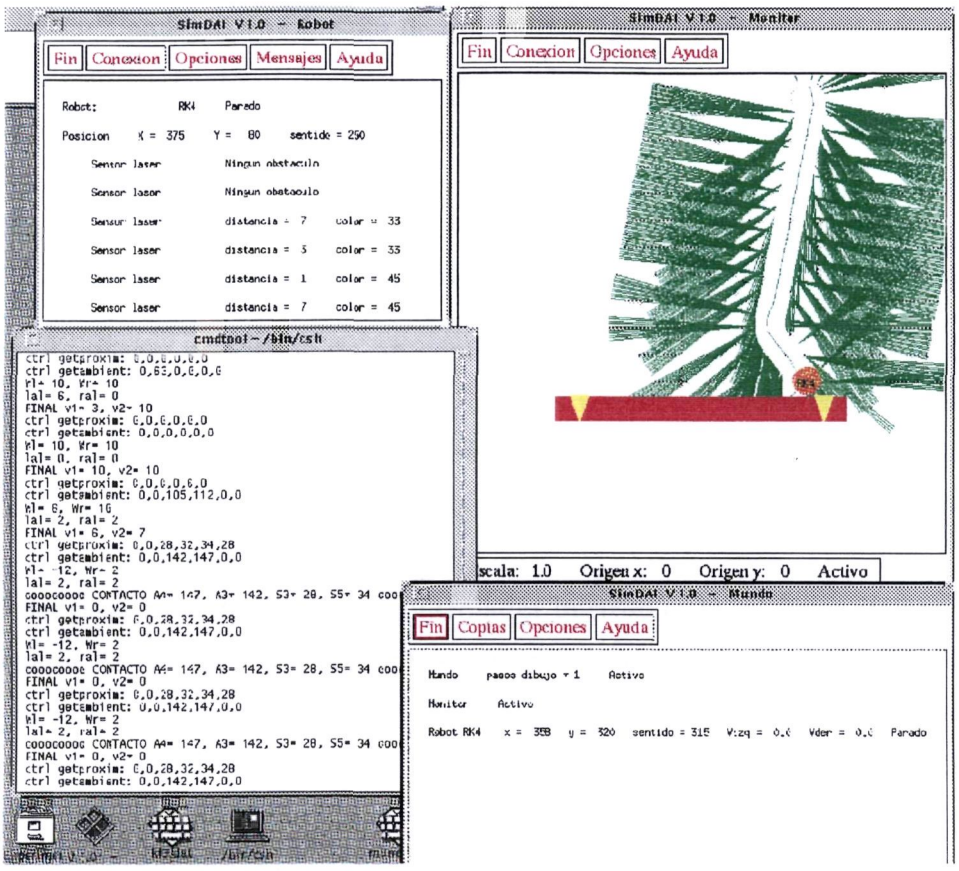

Figura 5.2: El entorno simulado.

prueba la idoneidad del simulador SimDAI para este tipo de experimentos.

5.2 Diseño de Habilidades para SimDAI

En el capítulo anterior se describió el funcionamiento del modelo utilizando como ejemplo precisamente este experimento. Allí se detallaba el funcionamiento interno del modelo pero no cómo se han construido las habilidades que lo implementan (`Empujar`, `Alinearse`, `Buscar_Caja` y `Evitar_Obstáculo`).

El modelo propuesto no establece ningún mecanismo para implementar el método *Ejecutar*. En la Figura 4.4 aparece el correspondiente a la habilidad `Empujar`, que simplemente consiste en avanzar en línea recta. Sin embargo, no todos son tan sencillos. En esta sección se va a mostrar cómo el modelo permite integrar sistemas de razonamiento complejos dentro de las habilidades.

En concreto, se va a presentar el diseño de controladores borrosos para el control del robot. En primer lugar se va a realizar un diseño heurístico, basado en la experiencia y a continuación se propone un método para generar dichos comportamientos de forma

automática, mediante un mecanismo de aprendizaje genético.

La razón principal para utilizar controladores borrosos es que constituyen una herramienta muy flexible y que han sido utilizados con éxito en múltiples sistemas autónomos [Saffiotti *et al.*, 1995].

5.2.1 Diseño Heurístico de Habilidades

El proceso de diseño heurístico de un controlador consta de varios pasos. En primer lugar, es necesaria la identificación de las entradas y salidas relevantes. A continuación habrá que describir estas variables de forma borrosa; es decir, detallando el conjunto de variables lingüísticas que se definen sobre ellas. Por último habrá que realizar el diseño de las reglas de control.

El primer paso en el diseño de un controlador borroso es la selección de una descripción adecuada de las entradas relevantes para el control, como puede ser la *distancia* a los obstáculos, de forma análoga a la que utilizaría una persona para su descripción. Así, dada una distancia numérica d_i a un obstáculo detectado por un sensor, D_i se define como el rango de todos los posibles valores de la distancia d_i. Con el fin de manejar la incertidumbre que subyace bajo la apariencia de entradas de los sensores (distorsionada después del proceso de adquisición), los valores numéricos de la distancia d_i se pueden traducir a etiquetas simbólicas cualitativas a través de un proceso de "borrosificación" [Zadeh, 1973], transformando la entrada de los sensores en variables lingüísticas como "cerca", "lejos", etc.

Una variable lingüística, según [Zadeh, 1973], es una variable cuyos valores son frases de un lenguaje natural o artificial, es decir, la concatenación de términos atómicos: etiquetas (adjetivos), modificadores (como "muy", "poco", "algo", etc.), la negación ("no") y las marcas (paréntesis).

El significado de una variable lingüística se define como un subconjunto borroso para el que el valor de la variable lingüística se utiliza como una etiqueta. Un subconjunto borroso A de un universo del discurso U viene dado por una función de pertenencia $\mu_A : U \rightarrow [0, 1]$ que relaciona cada elemento y de U con un número $\mu_A(y)$ que representa el grado de pertenencia de y a A.

La operación de "borrosificación" (que depende de la aplicación) produce como efecto la transformación de un conjunto no borroso o cantidad, en un conjunto borroso. Conviene destacar en este punto que definir el valor de, por ejemplo, la variable lingüística *distancia* como una etiqueta natural (como *cerca*) representa un significado

mucho menos preciso que su correspondiente valor numérico de cuántos centímetros hay hasta el obstáculo.

El Apéndice A describe con detalle las implicaciones que tienen estas definiciones. Allí se explican los conceptos básicos de la lógica borrosa y del proceso de razonamiento. En esta sección se detallará únicamente el proceso de diseño que hay que realizar.

La definición de las salidas se realiza empleando el mismo formalismo que para las entradas, es decir, definiendo las variables borrosas adecuadas. El último paso es el diseño de las reglas de control que harán uso de dichas variables. El diseño de estas reglas para el control de un sistema autónomo no es sencillo, ni siquiera en los casos más simples.

Considérese, por ejemplo, un robot real con ocho variables de entrada (sensores) y dos de salida (motores), para el que se quiere diseñar un controlador que le permita moverse por un entorno desconocido de forma aleatoria y sin colisiones. Si se codifica cada uno de los sensores de entrada con solo dos etiquetas lingüísticas (*Cerca, Lejos*) y cada una de las salidas con cinco etiquetas lingüísticas (*Adelante-Rápido, Adelante, Parado, Atrás, Atrás-Rápido*). El número de posibles reglas borrosas sería 6.400 ($2^8 * 5 * 5$). Pero si se utiliza un grado de granularidad más alto, por ejemplo, si se codifica utilizando cinco etiquetas lingüísticas para los sensores, el número de reglas pasa a ser 9.765.625. Utilizando una codificación de ocho etiquetas lingüísticas por variable, el número de reglas posibles llega al orden de los miles de millones (1.073.741.824).

Lógicamente, sólo se necesitan unas pocas reglas para obtener un comportamiento tan simple. El problema es cómo elegir las reglas. El método más comúnmente utilizado ha sido dejar el diseño de las reglas al programador. Sin embargo, los humanos suelen pensar de forma antropomórfica, lo que puede causar problemas.

Considérese, por ejemplo, una versión simplificada del robot autónomo anterior, que tenga sólo dos sensores de proximidad (*sensor*1 y *sensor*2) y dos motores (*motor*1 y *motor*2). Cuando se le pide a cualquier programador que escriba las reglas que le permitan deambular por el mundo sin chocar con nada, las reglas obtenidas son algo parecido a **Si** *sensor*1 **Es** *Cerca* **Entonces** *Motor*1 **Es** *Muy_Adelante*, lo que significa que cuando el sensor izquierdo percibe un obstáculo el motor izquierdo incrementa su velocidad para alejarse de él girando hacia la derecha. Análogamente, se definirá la regla simétrica para girar hacia la izquierda si se encuentra un obstáculo a la derecha.

Si se prueba este grupo de reglas en un robot, se comprueba que el robot comienza a incrementar su velocidad paulatinamente. El problema se puede definir como un

aumento continuo de la entropía. Se puede corregir fácilmente escribiendo las reglas de forma inversa - **IF** *sensor*1 **Es** *Cerca* **Entonces** *Motor*2 **Es** *Atras* - y añadiendo una nueva regla para incrementar la velocidad cuando no hay obstáculos.

Otro problema con las reglas diseñadas por humanos es que las diseñan para un mundo teórico: la mente humana. Esto significa que las reglas y las etiquetas lingüísticas se tienen que retocar muchas veces hasta alcanzar un comportamiento aceptable.

En resumen, el diseño heurístico de un controlador borroso no es un asunto trivial. En el Anexo A se muestra un ejemplo de controlador difuso diseñado para el problema descrito de evitar obstáculos. En [Matellán *et al.*, 1995b] se pueden encontrar las definiciones de algunos otros, aplicadas a un robot como el descrito.

5.2.2 Diseño de Habilidades mediante Aprendizaje

Dado que el diseño de reglas por métodos heurísticos es complejo, sería muy interesante disponer de métodos automáticos de generación de las mismos. Se pueden encontrar diversos métodos en la literatura. Éstos abarcan desde los métodos matemáticos [Steels, 1990] a las redes de neuronas artificiales [Maes and Brooks, 1990], pasando por los métodos simbólicos o los algoritmos genéticos [Koza, 1991]. En esta sección se va a describir un mecanismo basado en algoritmos genéticos para la generación de reglas borrosas. Concretamente se generará un controlador para el problema anterior.

En la literatura se pueden encontrar ejemplos del uso de métodos genéticos de aprendizaje de planes para sistemas autónomos. Por ejemplo, en [Cook, 1993], se presenta un método genético para formular nuevos conjuntos de reglas de decisión de bajo nivel para movimientos de robots. Cada regla comprueba que ciertas condiciones se cumplen (obstáculos detectados, posición de la meta, etc.), y ejecuta los operadores correspondientes (mover hacia delante, mover hacia atrás, girar a la derecha, girar a la izquierda). La selección genética tiene lugar entre conjuntos de reglas de comportamiento después de probar los planes.

Se pueden encontrar también trabajos relativos a la evolución genética de conjuntos de reglas así como metodologías para su generación [González *et al.*, 1994]. Por ejemplo, en [Thrift, 1993] se presenta un algoritmo genético para el diseño de las reglas borrosas capaces de centrar un carrito aplicando una fuerza simple. [Karr, 1991] presenta la aplicación de este método a tres sistemas físicos diferentes: un sistema para el control del nivel de un líquido, un sistema de control de pH y un sistema de

control de satélites. En cada una de estas aplicaciones, los controladores borrosos diseñados genéticamente mejoran los diseñados por los humanos. Sin embargo, se han hecho pocos experimentos sobre la evolución de las etiquetas lingüísticas borrosas. En [Lee and Takagi, 1993] se aplica un método genético al problema clásico de control del péndulo invertido, determinando cada función de pertenencia para un modelo borroso denominado TSK, el número de reglas borrosas y los parámetros de los consecuentes de las reglas.

También se han propuesto mecanismos distintos al que se va a describir para la evolución de controladores borrosos aplicados al control de vehículos autónomos. Por ejemplo, en [Cooper, 1995] se presenta un método general para la evolución de controladores borrosos basados en reglas. Otra metodología basada en una estructura jerárquica priorizada utilizando un algoritmo genético se aplica al control de un vehículo autónomo [Hoffmann and Pfister, 1994].

El objetivo de esta sección es comprobar si las reglas obtenidas se pueden emplear dentro de la arquitectura ABC^2 y si dichas reglas son capaces de controlar correctamente el robot autónomo. Para ello, se propone un método de aprendizaje genético [Goldberg and Holland, 1988] que partiendo de una serie de variables lingüísticas predefinidas como *Cerca, Lejos*, etc. para los sensores, *Despacio y Deprisa* para los motores, genere el conjunto de reglas adecuado.

El método propuesto consiste en codificar los antecedentes y consecuentes de las reglas borrosas en forma de "cromosoma", más concretamente las etiquetas lingüísticas. Después se forma una "generación" de individuos, es decir, un conjunto aleatorio de esos cromosomas, cada uno de los cuales representa un algoritmo de control diferente, y por último se prueban en el entorno.

Se define además una función de evaluación del rendimiento de los controladores, en función de sus colisiones, velocidad, etc. Con esa función se elige el mejor de los cromosomas y a partir de él se obtiene una nueva "generación". Para ello se utilizan los operadores tradicionales de la programación genética [Goldberg, 1989]: mutación y sobre-cruzamiento (*crossover*).

El proceso de aprendizaje se ha realizado tanto en el simulador SimDAI como con robots reales. En el caso del simulador se utilizó un entorno con obstáculos generados aleatoriamente. En los experimentos con el robot éste se colocó dentro de un entorno sencillo consistente en un área rectangular con obstáculos situados como indica la Figura 5.3. Las paredes eran de gomaespuma sobre la superficie de una mesa de

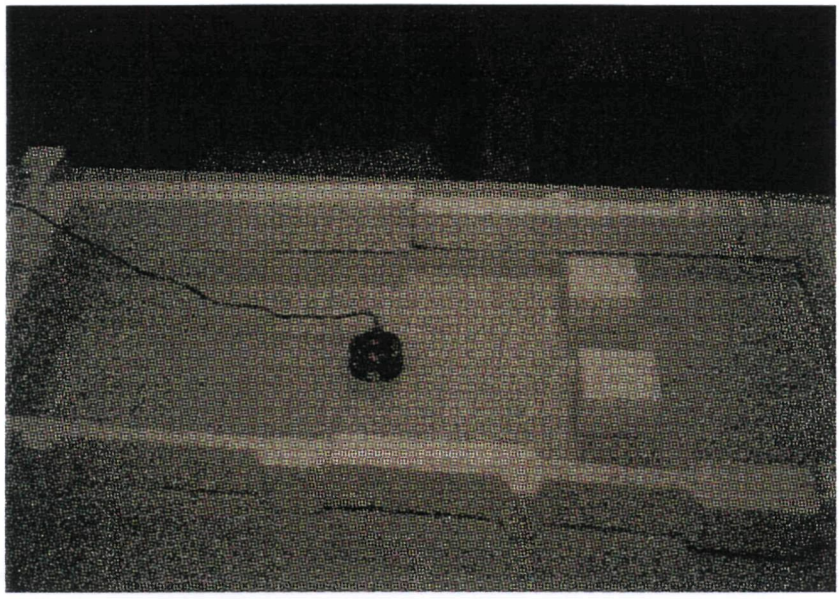

Figura 5.3: Entorno de Experimentación.

madera. La mesa estaba situada en un laboratorio iluminado siempre con luz artificial. El robot real tiene ocho sensores, pero para hacerlo equivalente al simulado de la sección anterior se han agrupado los 6 frontales en dos, izquierdo y derecho. Por tanto, el controlador borroso contará con 2 variables de entrada y 2 de salida (una para cada motor).

Se realizan cinco particiones borrosas de cada una de las cuatro variables (*sensor*1, *sensor*2, *motor*1 y *motor*2). Por tanto, las estrategias de control individuales se pueden definir con dos tablas de 5x5, una para cada motor. Cada tabla relaciona las funciones de pertenencia de cada uno de los sensores con las de un motor. Las tablas están codificadas como cromosomas con "alelos" $0, 1, 2, 3, 4, 5$ que corresponden a las funciones de pertenencia *Muy_Atrás, Atrás, Parado, Adelante, Muy_Adelante* y *Nulo*, donde el último indica que no hay ninguna relación entre las dos funciones de pertenencia.

Cada individuo tiene pues un cromosoma de longitud cincuenta. Los alelos de dicho cromosoma son las cinco funciones de pertenencia definidas sobre el dominio de los motores más el código nulo. El comportamiento se obtiene aplicando las operaciones habituales: fuzzificación, composición y defuzzificación por el método denominado "centro de gravedad" (consistente en calcular el valor numérico en función del centro de gravedad del trapecio que define la variable borrosa).

Resta definir el criterio de adecuación, denonimado en ingles *fitness* y que se representa mediante Θ, para evaluar el rendimiento de los distintos individuos. El criterio

elegido es función de tres variables medidas directamente sobre el robot Khepera, y otra más directamente sobre el conjunto de reglas:

$$\Theta = \frac{V(1-\sqrt{D})(1-I)}{reglas}$$

donde la variable I representa el valor normalizado del sensor que presenta el nivel más alto de activación: $I = \frac{sensor}{1023}$ (donde 1023 es el valor máximo que pueden devolver los sensores). V es la velocidad media de rotación de las dos ruedas: $V = \frac{media}{10}$ (donde se ha fijado 10 como valor máximo) y D es el valor absoluto normalizado de la diferencia entre la velocidad de las dos ruedas: $D = \frac{|v_1 - v_2|}{20}$ (siendo por tanto 20 la diferencia máxima entre las dos). De esta forma la función Θ se maximiza cuando el robot evita los obstáculos, mantiene una dirección más recta y no está parado, premiándose además el menor número reglas.

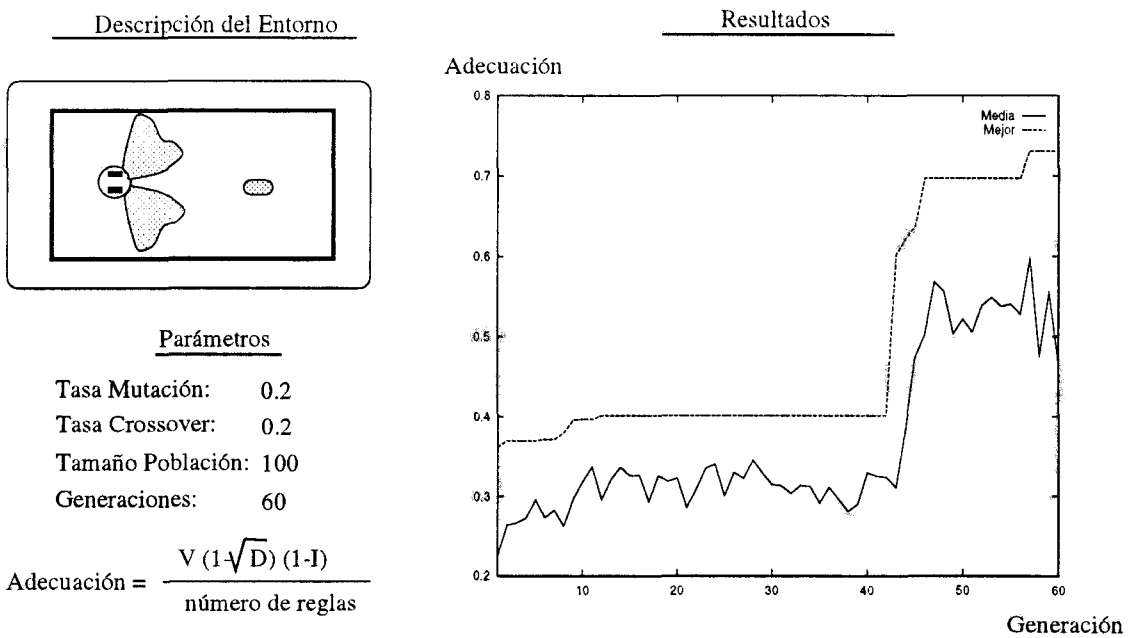

Figura 5.4: Comparación del *fitness* medio con el del mejor individuo.

La evolución del entrenamiento se realizó a través de un algoritmo genético estándar, que consistió en 100 generaciones de una población de 100 individuos. El operador de mutación se definió como la modificación de un código borroso en un nivel arriba o abajo o a un valor nulo. El operador de *crossover* se definió como un cruzamiento de dos puntos. El tiempo de vida de cada individuo se fijó en 30 segundos. Para la generación de la siguiente población se utilizó una estrategia *elitista*, lo que significa que el individuo más adaptado promociona automáticamente a la siguiente generación.

En la Figura 5.4 se puede apreciar la evolución de la adecuación media de cada generación y la adecuación del mejor individuo de cada generación. Como se puede comprobar en la definición de la fórmula, todos los valores están normalizados a uno.

La configuración de este experimento se muestra también en la figura, indicando que las probabilidades de uso de los operadores genéticos fueron Mutación: 0.2 y Sobrecruzamiento: 0.2. En la figura también se indica con un croquis el tipo de entorno empleado.

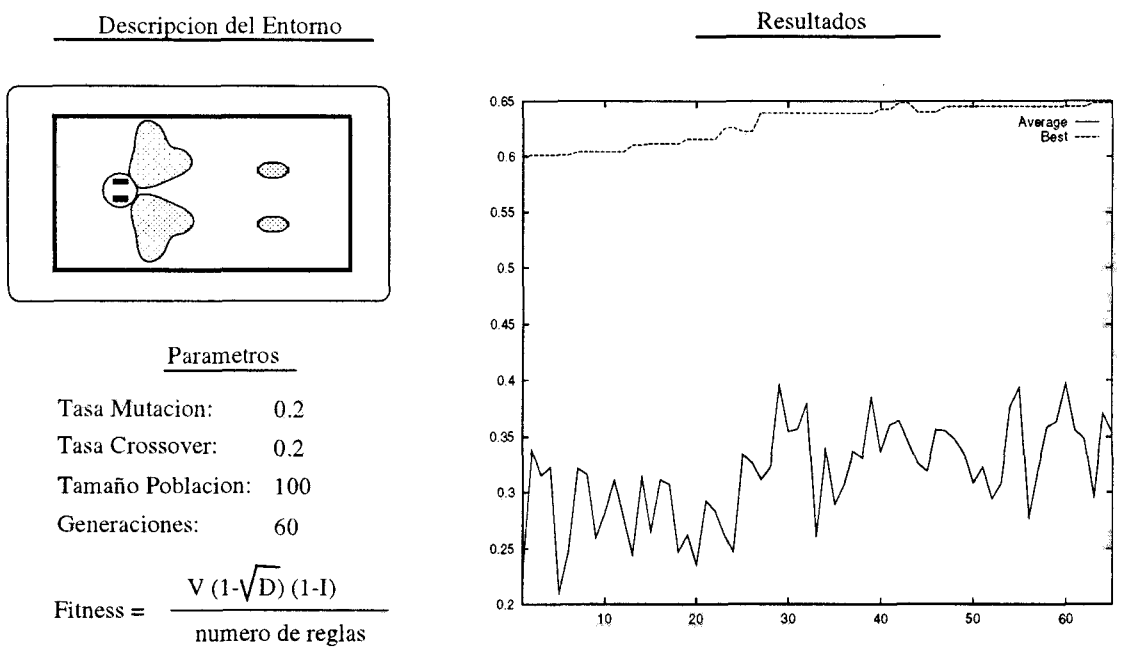

Figura 5.5: Comparación del *fitness* medio con el del mejor en un entorno con dos obstáculos.

La figura permite evaluar el proceso de aprendizaje. Así, se puede comprobar, que en la mayor parte de los experimentos, alrededor de la generación número 60 la población ha aprendido a evitar obstáculos.

La Figura 5.5 muestra el resultado del mismo experimento utilizando una configuración diferente. Esta figura muestra que si las probabilidades de mutación y *crossover* se incrementan, la velocidad del proceso de aprendizaje crece. También se han llevado a cabo experimentos incrementando sólo uno de estos factores y variando el número de individuos de cada generación y el número de generaciones.

También se ha probado a ponderar más un factor que los demás para mejorar alguno de los aspectos concretos de la habilidad, como puede ser "correr más deprisa" contra "mantenerse a salvo". Los detalles de todos estos experimentos pueden encontrarse en [Matellán *et al.*, 1995a].

El número medio de reglas de los individuos de las últimas generaciones muestra que este método es capaz de generar soluciones con un número reducido de reglas, que ha resultado ser menor de lo esperado, lo que constituye otro resultado interesante del experimento.

5.2.3 El Experimento sobre SimDAI

En las dos secciones anteriores se han mostrado dos métodos para diseñar las habilidades necesarias para poder llevar a cabo la tarea de empujar un objeto entre dos robots. El primer método empleaba la experiencia del diseñador para generar las reglas, mientras que el segundo la emplea para diseñar la función de evaluación y a partir de ella generar las reglas mediante técnicas basadas en algoritmos genéticos.

El conjunto de reglas borrosas generado para la habilidad `evitar_obstáculos` de forma heurística se puede analizar en la Figura A.2. Por su parte, un ejemplo de controlador borroso generado por medios genéticos es el que muestra la Figura 5.6.

En los experimentos realizados sobre el simulador SimDAI se han utilizado indistintamente los controladores generados genéticamente, por ejemplo para implementar el controlador *Ejecutar* de la habilidad `Evitar_Obstáculo`; y los definidos de forma heurística, el resto de las habilidades necesarias (`Alinearse`, ...).

Una vez definidas las habilidades de los robots, uno de ellos se inicializa con la meta empujar y el otro simplemente con la de evitar obstáculos. Los robots se mueven entonces por el entorno simulado hasta que uno de ellos detecta su punto de alineamiento. Para facilitar la tarea de reconocimiento, el punto de alineamiento tenía un color único en el entorno (en el experimento con los robots reales se utilizó una simplificación semejante, como se explica en las siguientes secciones).

Cuando el primer robot detecta el lugar, va hacia él, se alinea y le envía a su compañero la petición de alinearse. El otro robot la recibe, la acepta, y realiza el mismo proceso que el anterior. Una vez que él también ha conseguido alinearse envía el mensaje de información al otro robot y los dos comienzan a avanzar a la misma velocidad.

El funcionamiento del experimento, en lo que se refiere a la inserción de actos en la agenda, expansión de sus necesidades, elección de la habilidad a ejecutar, etc., es el explicado en la Sección 4.2.

En cuanto a la evaluación cualitativa del experimento, se puede concluir que el modelo es aplicable en este tipo de dominio, pues los experimentos realizados demostraron

que con un diseño correcto de las habilidades ABC^2 es capaz de controlar y coordinar a los robots para que realicen la tarea, siendo indiferente la configuración de obstáculos utilizada.

Además, el modelo cumple con las restricciones exigidas en cuanto a flexibilidad, puesto que es capaz de realizar su trabajo con configuraciones distintas del mundo simulado, incluyendo tanto distinto número de obstáculos y disposición de los mismos, como distintos puntos de partida.

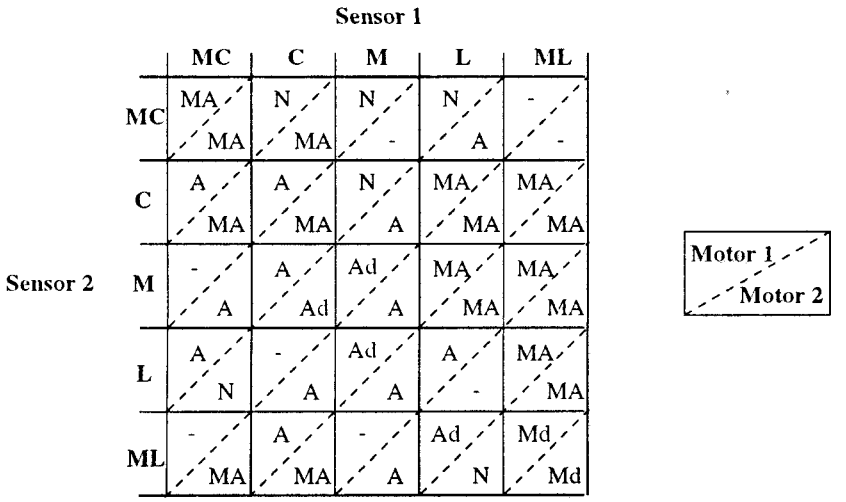

Figura 5.6: Controlador borroso generado genéticamente.

5.3 Experimentos con los Robot Reales

Como se explicó en la sección 5.1.1 otro de los objetivos que se perseguían al probar ABC^2 en este dominio, era mostrar su capacidad de adaptación tanto a entornos simulados como a entornos reales. En este sentido, se puede considerar que la sección anterior (haciendo hincapié en los aspectos de diseño de las habilidades) representa los experimentos realizados en un entorno simulado, en concreto SimDAI.

Para conocer el grado de reutilización que ofrece ABC^2 se replicó ese mismo experimento con robots reales utilizado el mini-robot Khepera, que se describe a continuación. Para ello fue necesario adaptar el problema como se describe en la Sección 5.3.2, obteniéndose los resultados comentados en la Sección 5.3.3.

5.3.1 El mini-robot Khepera y el Entorno Real

La plataforma utilizada para replicar el experimento con robots reales ha sido el mini-robot Khepera [Mondada *et al.*, 1993]. Este robot está basado en el microcontrolador

Motorola 68331, dispone de 256 Kbytes de memoria RAM y 256 Kbytes de memoria ROM. El micro-controlador se encarga de manejar todas las rutinas de Entrada / Salida y puede comunicarse mediante el puerto serie con una estación de trabajo. También se puede descargar el código en el robot pudiendo funcionar autónomamente gracias a sus baterías.

Se ha elegido esta plataforma por las siguientes razones:

- Su pequeño tamaño. (5.5 cm de diámetro)
- Su facilidad de control.
- Su relativo bajo costo.
- El hecho de que los experimentos puedan realizarse en una área reducida.
- En poco espacio se pueden emplear varios robots.
- Es fácilmente transportable (a otros laboratorios, conferencias, etc.).

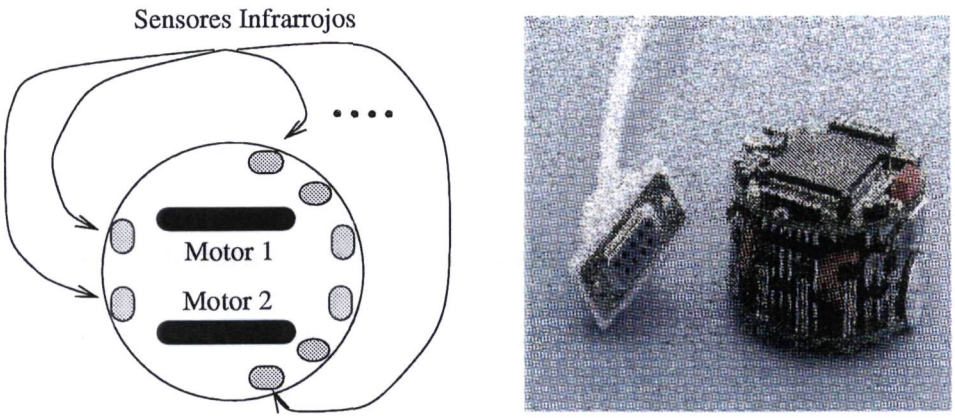

Figura 5.7: Distribución de los sensores.

Además, la placa base puede extenderse con placas adicionales entre las que se encuentran una cámara, un brazo, etc. La placa base, con la que se realizarán los experimentos, está formada por el micro-controlador, dos motores y los sensores. Los sensores de los que dispone la placa base son 8 sensores de infra-rojos, 6 situados en el frente y dos en la parte trasera (según la Figura 5.7), y dos odómetros, uno para cada rueda. Los sensores de infra-rojos miden tanto la proximidad a obstáculos como la luz ambiente. Los odómetros miden el desplazamiento de cada rueda.

Aunque este micro-robot puede funcionar de forma autónoma, debido a su escasa memoria (512 Kbytes), y a que el hecho de descargar el código hace difícil la depuración,

se ha utilizado conectado al puerto serie de una estación de trabajo. Por otra parte, el concepto de autonomía que se persigue es la ausencia de intervención humana y no el hecho concreto de tener todo descargado en el robot. Además, los robots necesitan comunicarse entre sí, lo que no es posible realizar más que a través de una estación de trabajo, por no disponer de módulos para la comunicación por radio.

Sin embargo, no es la comunicación la principal limitación de la placa base del Khepera. El principal problema es la poca cantidad de información que es posible extraer de los sensores de infrarrojos. Dichos sensores tienen un ángulo de apertura de 90 grados y una distancia útil máxima de unos 6 cm. Es decir, a la hora de detectar obstáculos, el sensor tiene en cuenta todo lo que hay en un sector de 90 grados, devolviendo únicamente un valor numérico entero entre 0 y 1023, obviamente poco significativo y además muy dependiente de las condiciones lumínicas del ambiente.

Los odómetros de las ruedas sufren una deficiencia de otro tipo: la acumulación de errores. A partir de estas medidas es muy difícil conocer la posición real del robot. Los giros, deslizamientos de las ruedas, colisiones, etc, hacen que los errores se acumulen de tal forma que transcurridos unos segundos de movimiento ya no es posible deducir la posición del robot a partir de los valores que hayan devuelto los odómetros.

Los problemas anteriormente citados hacen que a la hora de diseñar un experimento que utilice el micro-robot Khepera haya que tener en cuenta que:

- los robots no pueden saber su posición absoluta.
- los robots no pueden informar ni solicitar a otros robots su propia posición absoluta, ni la de ningún objeto del mundo
- para los robots es muy difícil reconocer objetos por su forma o color
- lo único que los Khepera son capaces de reconocer de manera fiable es la presencia de una luz

5.3.2 El experimento

El objetivo del experimento es el traslado de un objeto que necesita (supóngase que por razones de peso) la cooperación de dos robots. Debido a las limitaciones del robot a emplear, es necesario diseñar un entorno particular en el que se encuentren los dos mini-robots Khepera, el objeto a trasladar (marcado como *objeto* en la Figura 5.8) y varios obstáculos.

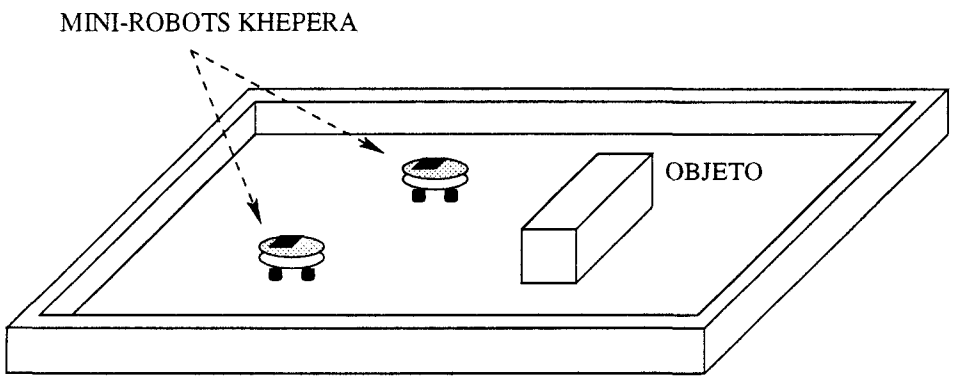

Figura 5.8: Entorno para el experimento de empujar.

La tarea puede dividirse en dos partes: prepararse para empujar y empujar. En las pruebas realizadas se apreció que la tarea realmente importante es la primera, poque las imprecisiones de los sensores de los robots reales hacen que los robots tarden mucho tiempo en conseguir la alineación deseada para que luego la acción de empujar sea efectiva. Esta primera tarea puede subdividirse a su vez en las siguientes subtareas:

1. El primer robot encuentra el objeto a empujar.
2. El robot se alinea con el objeto y envía al otro robot una descripción de su alineamiento.
3. El segundo robot trata de alinearse exactamente como el primero.

La primera fase se puede realizar mediante un controlador difuso sencillo, del tipo de los descritos previamente [Matellán *et al.*, 1995b], o mediante otro tipo de controladores. Por ejemplo, en este experimento se utilizó una versión del controlador reactivo propuesto por Valentino Braitenberg [Braitenberg, 1984] para alcanzar la posición correcta de alineación.

Una vez que el robot ha encontrado el objeto, se alinea con él de una determinada forma. Esta fase necesita un control más preciso que la fase anterior, pues el alineamiento determina si la tarea cooperativa puede ser realizada correctamente o no. Utilizando un robot real como el Khepera, la definición del alineamiento hay que realizarla en función de las medidas de los sensores.

En este sentido, se ha realizado la definición del alineamiento en función de variables borrosas definidas para cada sensor del robot Khepera. Así, se supone que un robot que usa un controlador borroso para alinearse se ha alineado cuando sus sensores de

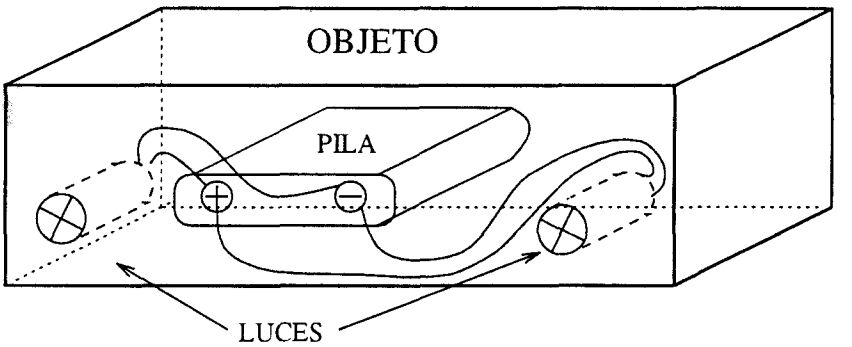

Figura 5.9: Diseño del objeto a transportar

proximidad y sus sensores de luz ambiente devuelven unos valores borrosos concretos. En ese momento el robot informa al otro de *cómo* se ha alineado.

Se han probado diversas alternativas de comunicación, consistentes en el intercambio de conceptos no-borrosos y borrosos. En el primer caso (usando conceptos no-borrosos) el emisor le dice al otro robot los valores exactos que han medido sus sensores. Esta solución sería la ideal si los dos robots fuesen físicamente idénticos, incluyendo la sensibilidad de los sensores, y si no se produjesen errores en las medidas.

En el intercambio borroso se han probado tres versiones: enviar la información completa de las variables borrosas, (lo que quiere decir que utilizarán exactamente las mismas); enviar sólo los valores de activación de etiquetas borrosas comunes aunque definidas de forma distinta en cada robot; y por último, emplear un conjunto de etiquetas borrosas independiente de las de los dos robots.

En el experimento con los robots reales el entorno se percibe mediante los sensores de luz ambiente y de proximidad. Dado que los sensores del Khepera son muy sensibles al entorno externo, se realizaron diversas pruebas preliminares para elegir una configuración significativa, ajustando las posiciones iniciales de los robots y la luz ambiente del laboratorio.

Así mismo, para que los robots puedan distinguir el objeto, éste se ha diseñado según el esquema de la Figura 5.9. Las luces son dos simples bombillas de 1.5 Voltios alimentadas por la correspondiente pila. El objeto en sí no es más que un bloque de gomaespuma azul con el peso suficiente para que un robot no pueda trasladarlo por sí solo, aunque como se analizará a continuación no haría falta garantizar mediante peso que un robot solo no puede trasladarla, pues el propio mecanismo de negociación impuesto a los robots hará que los dos empujen a la vez.

5.3.3 Resultados del experimento

En el experimento con los robots reales, se ha medido la distancia al objeto para saber cómo se utilizan los valores comunicados en el controlador. En algunas situaciones el tiempo empleado por el segundo robot para alinearse también es una información significativa. Por ejemplo, en el caso de utilizar el método tradicional no borroso, el segundo robot es incapaz de alinearse en un tiempo razonable. Ello se debe a que en el mundo real hay muy pocas probabilidades de que los sensores del segundo robot midan *exactamente* lo mismo que los correspondientes del primer robot.

Sin embargo, para el caso de los mecanismos difusos el tiempo empleado no es significativo para discriminar entre las distintas opciones. En la mayoría de los casos sólo mide la calidad del controlador y su tolerancia a las perturbaciones del entorno (sobre todo a las variaciones de luz ambiente).

Se ha realizado un número significativo de pruebas para cada tipo de comunicación (enviar variables borrosas completas, valores de activación de etiquetas comunes o utilizando un protocolo independiente). Para ello se partió de una posición fija del primer robot: alineado enfrente de la luz exactamente a 15mm. Este robot enviaba su percepción borrosa de esta situación al segundo que se alineaba en función de ella.

Protocolo de comunicación	Distancia Media
Variables borrosas completas	15.125 mm
Activación de etiquetas comunes	19.375 mm
Protocolo independiente	15.375 mm

Tabla 5.1: Distancias medias según el protocolo de comunicación borroso

Después de 50 pruebas se obtuvieron las distancias medias reflejadas en la Tabla 5.1. Según estas medidas, el primer método es el más exacto, es decir, es el que alinea al segundo robot de forma más parecida a como se alineó el primer robot. Este resultado era predecible, pues es el método que pasa más información al controlador.

El segundo es el menos exacto en el alineamiento ya que las etiquetas se definen de forma distinta sucediendo por ejemplo, que una distancia realmente próxima resulte mayor que una comunicada como lejana. El tercer método tiene un rendimiento similar al primero. Ello es debido a que los dos robots comparten la misma interpretación de un concepto. Este método presenta un buen rendimiento, sobre todo si se tiene en cuenta que es más tolerante al ruido externo. Además, este método es el que utiliza menos recursos de comunicación, puesto que sólo necesita intercambiar un concepto. Una

discusión más extensa de este protocolo puede encontrarse en [Matellán *et al.*, 1996].

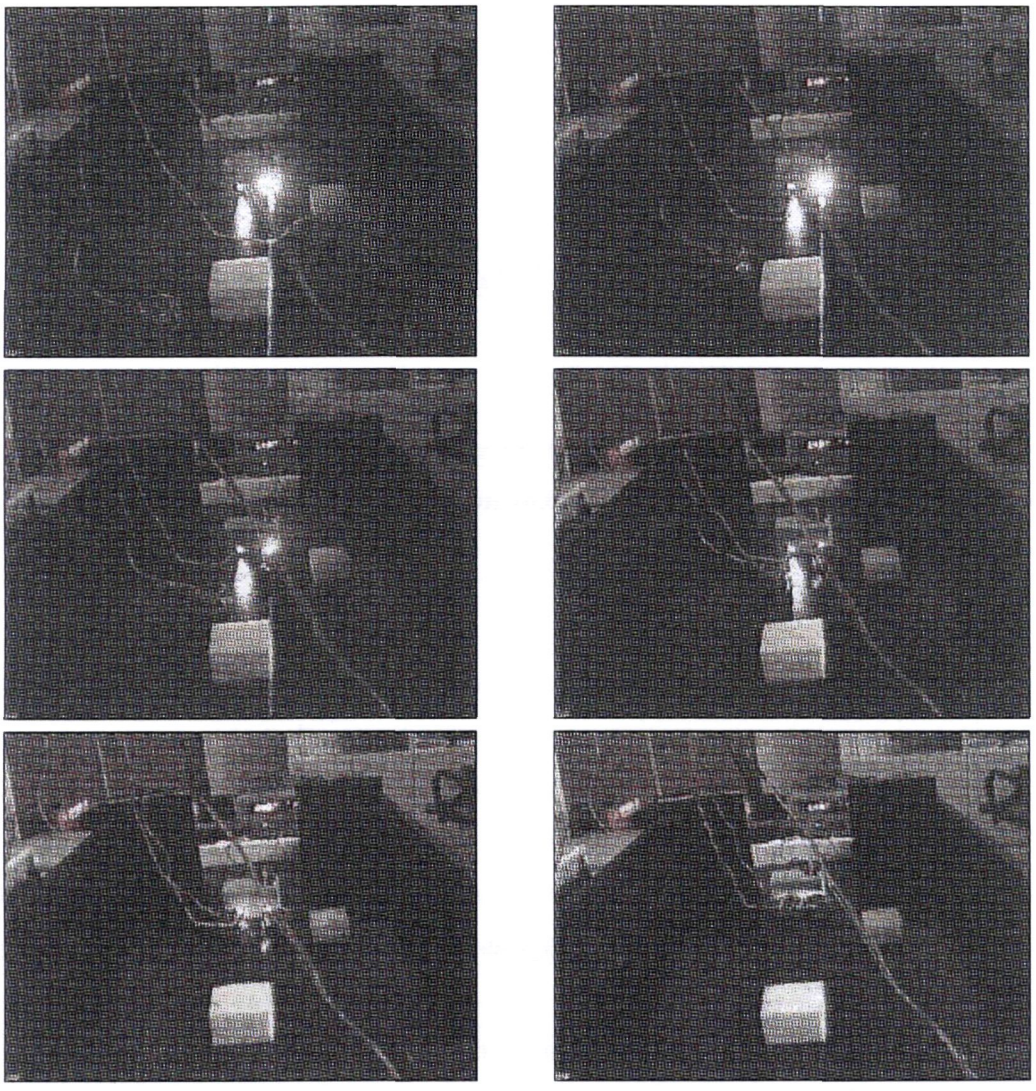

Figura 5.10: Secuencia de un experimento con mini-robots Khepera.

Las imágenes de la Figura 5.10 muestran una secuencia de imágenes correspondiente a uno de los experimentos realizados. La secuencia se sigue en el orden normal de lectura (de izquierda a derecha y de arriba hacia abajo). Puede verse como los robots utilizan las habilidades para evitar obstáculos, localizar el objeto por la luz que emite, alinearse con él y finalmente empujarlo.

En la descripción anterior se ha hecho especial hincapié en el problema de la comunicación de la información de alineamiento porque durante las pruebas demostró ser el punto crítico. Por lo que respecta al resto de las habilidades y al funcionamiento del modelo, como en el caso del simulador, se ajustó a lo expuesto en la Sección 5.3.2.

Capítulo 6

ABC^2 en la RoboCup

The World Cup Robot Soccer is an attempt to promote AI and robotics research by providing a common task, soccer, for evaluation of theories

Hiroaki Kitano

El año 1997 pasará a la historia, entre otras muchas cosas, por ser el año en el que por primera vez un ordenador consiguió ganar en un torneo uno contra uno al campeón mundial de ajedrez Gary Kasparov. Este era un objetivo largamente perseguido. De hecho se consideraba como uno de los problemas paradigmáticos dentro de la IA. Por otro lado, puede entenderse como el final de una época, la de resolución de problemas simbólicos (de juguete, para algunos investigadores) y el comienzo de otra nueva, más cercana a los problemas del mundo real.

El comienzo de esta nueva época en investigación en IA implica la necesidad de buscar nuevos retos en forma de un nuevo problema en el que la IA pueda comprobar sus avances. La experiencia del uso de juegos parece satisfactoria: permite comparar unas propuestas con otras, le da cierto interés, etc. Ello ha hecho que se generalicen las competiciones también entre los robots. Así, uno de los problemas propuestos como nuevo banco de pruebas (*benchmark*) es el fútbol entre robots.

Este capítulo describe los experimentos realizados empleando ABC^2 en este dominio. La primera sección se dedica a describir en detalle el dominio de la RoboCup, en particular su historia y algunos problemas concretos. La Sección 6.2 especifica las habilidades realizadas para este dominio y la forma de definir los jugadores. La Sección 6.3 comenta los resultados obtenidos. Por último, la Sección 6.4 describe los primeros pasos para conseguir un equipo formado por robots reales Khepera controlados mediante ABC^2.

6.1 El Dominio de la RoboCup

Diversos investigadores, fundamentalmente japoneses y norteamericanos, han propuesto una plataforma común para la prueba de diferentes tecnologías, que incluye el diseño de agentes autónomos (robots), la colaboración multi-agente, el razonamiento en tiempo real, la integración de sensores, etc. Esa plataforma de pruebas consiste en el desarrollo de un partido de fútbol entre dos equipos de robots autónomos y se conoce como *RoboCup*. Esta sección presenta los orígenes, promotores y objetivos de esta competición y describe alguno de los retos concretos que plantea.

6.1.1 Historia de la RoboCup

La RoboCup constituye un intento de promocionar los campos de la robótica y la inteligencia artificial proporcionando un problema estándar, el fútbol, donde se pueden integrar gran cantidad de tecnologías. Este dominio abarca, por ejemplo, los principios de diseño de agentes inteligentes (como es el caso de ABC^2), la colaboración multi-agente, el razonamiento en tiempo real, la construcción de robots, la fusión de información generada por sensores distribuidos, la identificación de estrategias, etc. Para ello se define la RoboCup como una tarea para un equipo de múltiples robots capaces de moverse rápidamente.

Actualmente la RoboCup consta de las siguientes ligas:

- *Simulator League*: Competición basada en el uso de simuladores.
- *Small Robot League*: donde se utilizan los robots de la clase Formula 180 (F-180) en equipos de 5 robots.
- *Full Set Small Robot League*: competición de equipos completos, esto es, 11 robots por equipo, de la clase Formula 180 (F-180).
- *Middle Size Robot League*: competición entre robots formados por cinco robots de la categoría Formula 2000 (F-2000).
- *Legged Robot League*: competición de robots que utilizan patas para su movimiento. Comenzará a modo de exhibición en la RoboCup de 1998.
- *Expert Robot League*: dedicada a evaluar el rendimiento de los robots en habilidades concretas.

En el futuro está previsto añadir a los juegos de la RoboCup:

- *Humanoid League*: en la que competirán robots con características humanas, catalogados como robots de Formula Uno: F-1.
- *TeleOperation Track*: en la que competirán robots de la categoría F-1 pero que pueden ser teleoperados.

La idea de robots que jugasen al fútbol fue propuesta por primera vez por el profesor Alan Mackworth de la universidad de British Columbia, (Canadá) en 1992 [Macworth, 1993]. El proyecto realizado se denominó *Dynamo* [Sahota *et al.*, 1995] y ha sido fuente de inspiración para la RoboCup. Hoy en día el profesor Mackworth sigue trabajando en temas relacionados utilizando su robot Spinoza.

Independientemente, un grupo de investigadores japoneses organizaron un *Workshop* denominado "Grand Challenge in Artificial Intelligence" en Tokio en Octubre de 1992 para discutir posibles problemas que sirvieran como test para la Inteligencia Artificial. En Junio de 1993 se propuso seriamente utilizar el fútbol para promocionar la ciencia y la tecnología estableciéndose para ello un proyecto denominado originalmente J-League[1] en el que participaban los investigadores Minoru Asada, Yasuo Kuniyoshi e Hiroaki Kitano. Debido al interés despertado fuera de Japón se decidió extenderlo en forma de proyecto internacional, al que se denominó *Robot World Cup Initiative*, abreviadamente RoboCup. El *ElectroTechnical Laboratory* (ETL), un centro de investigación perteneciente al gobierno japonés, comenzó entonces la investigación en sistemas multi-agentes utilizando el fútbol, e inició el desarrollo de un simulador de partidos de fútbol.

En Septiembre de 1993 se realizó el primer anuncio de la iniciativa y comenzó la redacción de las reglas específicas y la preparación de los aspectos técnicos y organizativos. Al mismo tiempo, varios proyectos relacionados ya habían comenzado, como el grupo del profesor Minoru Asada en la Universidad de Osaka o el de la profesora Manuela Veloso en la Universidad Carnegie Mellon.

En la *National Conference on Artificial Intelligence* (AAAI-94) celebrada en Seattle (Estados Unidos) en 1994, se llevó a cabo una discusión más detallada sobre la iniciativa de la RoboCup, incluyendo investigadores de los Estados Unidos, Europa y Japón. El mismo año, el ETL anunciaba la primera versión, de *Soccer Server*, el simulador oficial de la RoboCup. realizada en LISP y denominada versión 0.

[1] Es el nombre de la liga de fútbol profesional recientemente establecida en Japón.

Durante la *International Joint Conference on Artificial Intelligence (IJCAI-95)* celebrada en Montreal (Canadá) en Agosto de 1995, se anunció la celebración de los primeros **Robot World Cup Soccer Games and Conferences** a celebrar durante el IJCAI-97 en Nagoya (Japón). Al mismo tiempo se decidió celebrar una Pre-RoboCup-96 para identificar los problemas potenciales de organizar la RoboCup a gran escala. En la misma conferencia se hizo la primera demostración pública del Soccer Server Version 1.0 (realizada en C++) y que desde entonces se distribuye a través del servidor WWW del ETL [2].

La Pre-RoboCup-96 se celebró durante la *International Conference on Intelligence Robotics and Systems* (IROS-96) celebrada del 4 al 8 de Noviembre de 1996 en Osaka (Japón). Compitieron 8 equipos en la *Simulation League* y se realizó una demostración de robots reales de la *Middle Size League.*

Los primeros juegos oficiales y conferencias de la RoboCup se celebraron en 1997, con la participación de 40 equipos (entre la competición de robots reales y simulados), atrayendo a más de 5000 espectadores en directo. El autor participó en esta edición inagural en la *simulation league*, con un equipo cuyos jugadores estuvieron controlados mediante ABC^2. La comunidad de la RoboCup está creciendo rápidamente y se espera que la RoboCup-98, a celebrar en Julio de 1998, congregue a más de 100 equipos, convirtiéndose en el evento de robótica móvil más importante de la historia.

6.1.2 Problemas Específicos del Dominio de la RoboCup

Cada dominio tiene sus particularidades. Por ejemplo, el experimento analizado en el capítulo anterior, debía afrontar los problemas que plantean los sensores del Khepera. En el caso de la *simulation league* de RoboCup estos problemas tienen que ver, entre otras cosas con la adaptación al entorno: un simulador cerrado, en el sentido de tener que adaptarse a sus restricciones; el tratamiento de la información, en su mayoría muy perecedera; los problemas de trabajar en un entorno competitivo, etc. Esta sección intenta describir algunos de esos problemas.

El simulador SoccerServer

Dentro del término "simulador oficial de la RoboCup" se engloban realmente varios programas desarrollados en el ETL por el equipo dirigido por Itsuki Noda [Noda, 1995]. El simulador propiamente dicho se denomina **soccerserver** y básicamente consiste en

[2] http://ci.etl.go.jp/noda/~soccer/server.html

un servidor escrito en C++ que acepta conexiones de "clientes" mediante *sockets* UDP. Cada uno de esos clientes representa un jugador, el cual debe especificar en la conexión a qué equipo pertenece. El programa, siguiendo las reglas tradicionales del fútbol, acepta un máximo de 2 equipos y de 11 jugadores por equipo, aunque permite jugar con cualquier número inferior de ambos valores.

La misión del simulador es gestionar el partido. Para ello acepta órdenes de los jugadores: girar (`turn <ángulo>`), avanzar (`dash <fuerza>`), chutar (`kick <ángulo, fuerza>`) y hablar (`say <mensaje>`); y les envía información como si la recibieran de sus sensores: ver (`see`) y oir (`hear`). Toda la comunicación se realiza utilizando cadenas de caracteres codificadas en ASCII.

Ángulo / Calidad:	Wide (180°)	Normal (90°)	Narrow (45°)
Normal	300 ms.	150 ms.	75 ms.
High	600 ms.	300 ms.	150 ms.

Tabla 6.1: Frecuencia de las informaciones en el simulador RoboCup

El jugador puede enviar una orden siempre que quiera, aunque el servidor sólo aceptará como máximo una orden cada 3 ciclos de información visual, siendo la duración del ciclo de información la mostrada en la Tabla 6.1.

El servidor envía a los clientes la información visual y auditiva de forma síncrona, aunque la frecuencia puede variar según el modo de visión que esté utilizando cada jugador y que es modificable por él. La Tabla 6.1 muestra la frecuencia con la que el agente recibe la información según el modo de visión — en modo `Normal` sólo informa de direcciones, no de distancias — y del ángulo con que desea ver.

Otro de los componentes del simulador es el programa **soccermonitor** que permite visualizar los partidos. La interfaz de este programa es el que se muestra en la Figura 6.1. En el centro aparece el campo con los jugadores representados con distintos colores. En la parte superior, los goles marcados por cada equipo y el tiempo transcurrido, que en la competición oficial son 3000 ciclos, equivalentes a 5 minutos de tiempo real. Y en la parte inferior aparecen, en la parte izquierda, los mensajes generados por el *soccerserver* y en la parte derecha los mensajes que el simulador recibe de los jugadores (`dash`, `turn`, `kick`, `say`).

Una tercera herramienta permite realizar grabaciones de los partidos y su posterior visualización. El nombre de este programa es `logplayer` y su interfaz es similar al de cualquier reproductor de vídeo o CD, con opciones para detener la grabación, avanzar,

Figura 6.1: El monitor de partidos de la RoboCup

retroceder, etc.

Para adaptar la implementación de ABC^2 a este simulador, cada uno de los jugadores es un proceso independiente. Cada uno de ellos controlado mediante ABC^2. Puesto que tanto la comunicación entre los miembros del equipo como la del propio jugador con sus sensores y actuadores se realiza a través del servidor, hubo que construir un módulo especializado de comunicaciones implementado en GNU-C++ sobre la interfaz para *sockets* UDP que proporciona el sistema operativo.

Este módulo permite la conexión con el servidor, tanto para la recepción de información, distinguiendo los mensajes de otros compañeros de los mensajes de los sensores, como para el envío de los comandos y mensajes generados por el agente. Además hubo que diseñar la estructura adecuada para almacenar la información relevante y crear las funciones necesarias para realizar la interpretación de los mensajes recibidos del

simulador.

Incertidumbre en la Información

Cada uno de los robots recibe del simulador la información que correspondería a sus sensores. Concretamente, recibe datos equivalentes a información visual y auditiva. La información se recibe en forma de mensajes ASCII, con el formato mostrado en la Formalización 6.1. La información correspondiente al ángulo y distancia a los objetos en los mensajes `see` (visuales) se reciben de forma relativa a la posición del jugador. Además, dicha información no es fiable, siendo más inexacta cuanto mayor es la distancia al elemento. El error se genera mediante una función de ruido aleatorio en función de la distancia. Por supuesto, el jugador sólo recibe información de aquellos objetos que caen dentro del rango de visión definido según la Tabla 6.1.

Formalización 6.1 Estructura de los mensajes del simulador

(see <Tiempo> $Objeto_1$ $Objeto_2$ ... $Objeto_n$)
$Objeto_i$ ::= ($Nombre_i$ *Distancia Dirección CambioDist CambioDir*)
$Nombre_i$ ::= (player <Equipo> <Número>)
(goal <Lado>)
(ball)
(line [l|r|c|t|b])
(flag [l|c|r] [t|b])
(flag p [l|r] [t|c|b])

(hear <Tiempo> <Dirección> <Mensaje>)

Los objetos que un jugador puede ver son:

Jugadores que comienzan con la cadena `player` y que puede incluir información acerca del equipo al que pertenece el jugador y su número. Todo ello si está suficientemente cerca, porque si está lejos se omite el número y si lo está mucho, también el equipo.

Porterías representadas por `(goal <Lado>)`, es decir, `(goal r)` corresponde a la portería derecha y `(goal l)` a la izquierda, según indica la Figura 6.2.

Bola descrita como `ball`.

Líneas del campo, donde la *Distancia* se mide sobre la recta correspondiente al centro del campo de visión del agente y *Dirección* es el ángulo que forma dicha recta con la línea en cuestión. Las letras corresponden a los bordes del campo (izquierdo `l`, derecho `r`, superior `t`, inferior `b` y la línea central `c`).

Banderas (`flag`), con las que el jugador puede tener una idea de su situación en el campo. Las banderas existentes son las que se muestran en la Figura 6.2, junto con las coordenadas a las que corresponderían para un equipo que juegue en la parte izquierda del campo.

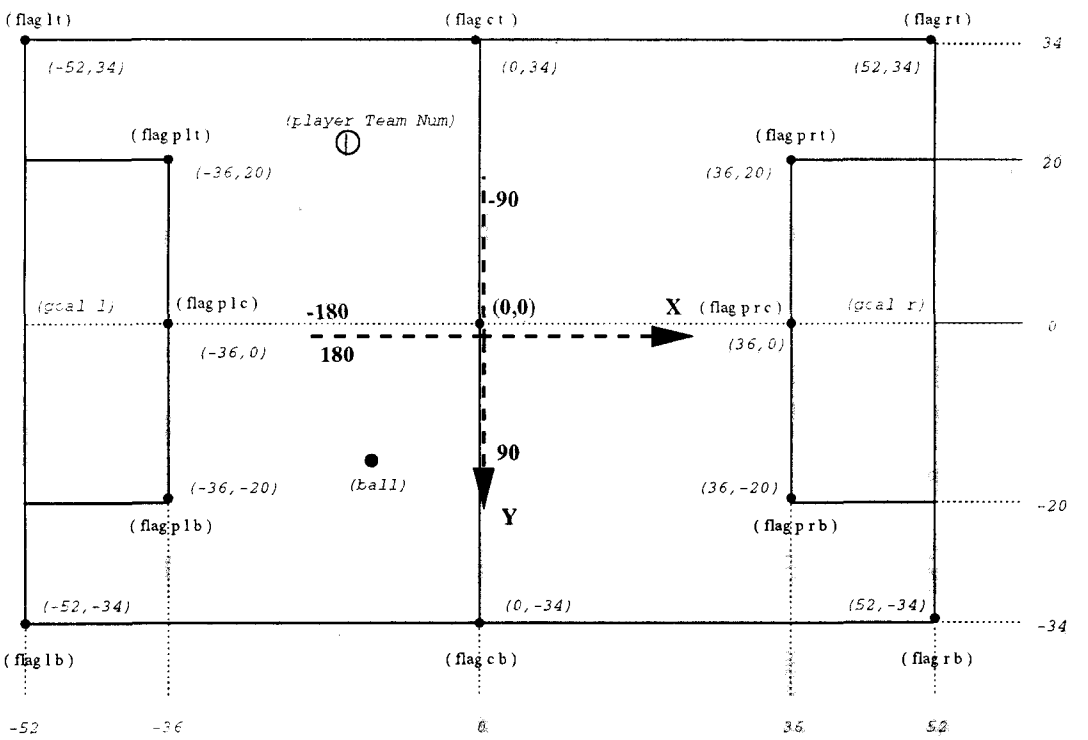

Figura 6.2: Informaciones proporcionadas por el simulador de la RoboCup.

Como consecuencia de esta implementación de los "sensores", el jugador no conoce su posición absoluta en el campo. Sólo recibe información de lo que ve desde ella. Desde luego, para poder jugar al fútbol con un mínimo rendimiento, es necesario que el jugador conozca su posición. Para lograrlo hubo que implementar un analizador (*parser*) que realizase un análisis de la información recibida. Dicho analizador es capaz de calcular la posición en la que se encuentra el jugador en un determinado instante, si tiene a la vista al menos tres elementos de posición conocida, de los que se reflejan en la Figura 6.2. También funciona con sólo dos si uno de ellos es una línea. En el resto

de los casos, con dos elementos existen, en general, al menos dos posibles posiciones, y con un solo elemento éstas son infinitas.

La posición calculada tiene cierto grado de incertidumbre, debido a que los sensores proporcionan datos inexactos. Por ello, muchas de las habilidades se definen con un cierto grado de "tolerancia" que se puede fijar en la inicialización de los jugadores.

Las acciones del robot también son inexactas. Es decir, si el robot decide que quiere girar un determinado ángulo, el giro producido realmente tiene cierto grado de error. Del mismo modo, los desplazamientos de la bola (comando `kick`) y los movimientos (`dash`) tienen cierto grado de error que el simulador calcula para cada uno de ellos.

Lo mismo ocurre con el resto de informaciones que mantiene el robot. Son informaciones inexactas por el error al que inducen los sensores. Además, dicha información se degrada con el tiempo, por lo que se almacena además el instante en que se recibió la última información sensorial de las mismas. Por ejemplo, se puede haber visto que un compañero se encontraba en una determinada posición, pero si por alguna razón sale del campo de visión, a medida que pasa el tiempo la certeza sobre su posición real irá disminuyendo.

Necesidad de Comunicaciones

En el dominio existe la posibilidad de que los jugadores se puedan comunicar. Para ello el simulador establece un canal de comunicación común para todos los jugadores de todos los equipos. Esta restricción podría ser fácilmente vulnerada, estableciendo por ejemplo canales directos de comunicación entre los jugadores, o incluso un planificador central que guiase sus acciones. Las normas de la RoboCup lo prohiben. Sin embargo, no ofrecen ningún mecanismo para detectar dichas acciones, dejando su cumplimiento, como establecen las propias normas, a un "pacto entre caballeros".

La forma de utilizar el canal común de comunicaciones es mediante la acción **say** que permite que un jugador envíe un mensaje a ese canal. Sin embargo, el hecho de enviar un mensaje no implica que el destinatario al que pretendía enviarlo lo reciba. El canal que proporciona el simulador tiene ciertas restricciones que lo hacen muy pobre desde el punto de vista de la cooperación:

- El número de mensajes que un jugador puede emitir está limitado a uno por ciclo de simulación.
- El número de mensajes que puede recibir está limitado a uno por cada dos ciclos.

Si hubiese más recibirá el primero que se emitiese.

- La longitud máxima son 256 bytes.
- Los mensajes se reciben sólo si el emisor y el receptor están en un cierto rango de distancias (50 m en la configuración oficial).
- Los mensajes se difunden a todos los jugadores, incluidos los del equipo contrario.

En el fútbol existen múltiples situaciones en las que la comunicación explícita entre los jugadores puede resultar provechosa. Por ejemplo, la Figura 6.3 muestra la versión del clásico problema del mínimo/máximo local en el mundo del fútbol. El jugador número 1, únicamente con su información visual decidiría que el mejor pase posible en su camino hacia la portería contraria es hacia el jugador número 2. Sin embargo, con una visión global del campo, como la que tiene un espectador, es mucho más razonable pasar al número 3 para que éste a su vez pase al número 4.

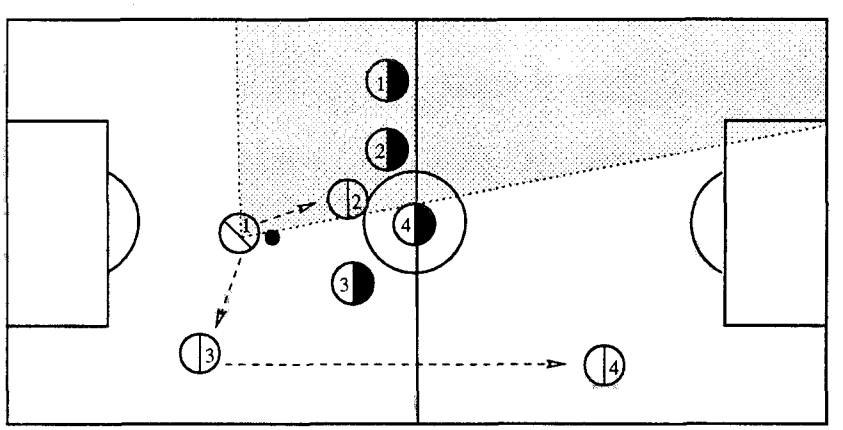

Figura 6.3: Problema del mínimo local en la RoboCup.

Por eso ese problema se puede definir como de máximo local: con la información local del robot 1 la decisión de pasar al 2 maximizaría el avance hacia la posición contraria. Sin embargo, con información global, la acción que maximiza el avance es pasar al número 3 y que éste pase al 4.

La única forma para conseguir que el jugador 1 decida la acción más correcta es proporcionarle más información. Para ello se puede hacer que el jugador mantenga información de lo que hubiese visto antes, por ejemplo antes de girarse hacia la izquierda. Como se ha mostrado en la sección anterior, dicho supuesto está contemplado,

pero debe tenerse en cuenta que la información es imprecisa y además la imprecisión aumenta a medida que pasa el tiempo.

Otra solución es utilizar la comunicación. El jugador número 3 puede pedir al número 1 explícitamente que le pase la bola. Si la heurística que rige el funcionamiento del número 1 da prioridad a las peticiones, entonces le pasaría la bola.

La comunicación en este entorno, a parte de la inseguridad en la entrega de los mensajes, agrava el problema de la degradación de la información. Si un jugador tiene que indicar, por ejemplo, la posición de otro a un tercero, cuando los dos interesados conocen su propia posición de forma inexacta, el error se acumula.

Seguridad en las comunicaciones

El entorno de la RoboCup se puede definir como un entorno hostil. Es una competición donde el contrario puede intentar ganar por todos los medios posibles. Entre otros, existe por ejemplo la posibilidad de intentar utilizar las comunicaciones del equipo contrario en beneficio propio o simplemente de sabotearlas.

La primera posibilidad sería intentar averiguar qué información se envía dentro de los mensajes. De conseguirse, un equipo podría adelantarse a las acciones del otro. Para evitarlo bastaría con cifrar de forma segura el mensaje.

Otra posibilidad es generar mensajes falsos. El objetivo en este caso sería engañar al contrario para hacerle cometer errores. Por ejemplo, solicitando un pase a un jugador. En este caso no basta sólo con cifrar las comunicaciones, hay que asegurarse de que el emisor es realmente quien dice ser (autenticación). Una forma de conseguirlo es utilizar mecanismos de clave pública [Ribagorda *et al.*, 1996] que permitan garantizar que un mensaje es enviado por quien dice ser y que sólo lo puede leer su destinatario.

Aún en ese supuesto existe el problema de la simple copia de mensajes, táctica que empleó algún equipo en la Pre-RoboCup96. Basta con copiar un mensaje, ya cifrado y por tanto válido, y re-emitirlo pasado cierto tiempo. Para evitar este tipo de tácticas se podría optar por numerar los mensajes, descartando simplemente aquellos que lleguen fuera de orden. Otra solución sería encriptar en el mensaje el momento de su emisión. Una solución de este tipo es la propuesta en [Stone and Veloso, 1998], aunque es más una propuesta para la competición del año 98, para la que se está planteando un nuevo formato de comunicaciones.

En cualquier caso, el coste (medido en tiempo de cifrado y descifrado de los mensajes) de implementar un mecanismo seguro de transferencia de mensajes y las limita-

ciones del sistema de comunicación, especialmente el reducido tamaño de los mensajes, imposibilitaba la utilización de mecanismos estándar de cifrado. Además de la extremada falta de fiabilidad de las comunicaciones, considérese que si todos los jugadores deciden enviar un mensaje en un ciclo y vuelven a hacerlo en el siguiente se emiten 44 mensajes, de los que sólo se recibe uno. Todo ello hizo que se considerase poco útil el uso de las comunicaciones durante la competición oficial de la RoboCup97, aunque sí se han utilizado en experimentos como el descrito en el Anexo B. De hecho, todos esos factores han hecho que de cara a la próxima competición se esté estudiando modificar la forma de realizar las comunicaciones.

6.2 Definición de los Jugadores

La definición de un agente según el modelo ABC^2 consiste fundamentalmente en la definición de las habilidades y de las heurísticas que controlan la forma en que éstas se combinan. Además de ello, es necesario especificar cuáles son los datos que necesitan tanto unas como otras para poder utilizarse. En ejemplos anteriores las informaciones, debido a la simplicidad de la tarea a realizar, se tomaban directamente del entorno. En el caso del dominio de la RoboCup, que es un entorno con mucha más información, es importante especificar el conjunto de informaciones.

Esta sección muestra ambas facetas. En primer lugar se definen las habilidades desarrolladas para la RoboCup y a continuación se especifica la forma y contenidos de la información que guarda cada agente.

6.2.1 Habilidades de la RoboCup

La parte más importante de la definición de un agente en ABC^2 es la definición de sus habilidades. En esta sección se describe cada una de las implementadas, describiendo para ello el método `Preparado`, que indica cuáles son las precondiciones para que se pueda ejecutar esa habilidad, y el método `Ejecutar`, que indica cuáles son las acciones que se ejecutan cuando se selecciona en la agenda. No se ha incluido la lista de necesidades de cada habilidad porque ésta se indica en la definición del agente.

La siguiente lista es una descripción de algunas de las habilidades empleadas por el equipo construido para la competición de la RoboCup 97 [Matellán and Borrajo, 1997]. Es una versión reducida y simplificada en su descripción, proporcionada para dar una idea del tipo de habilidades implementadas.

- Buscar_Bola:

  Preparado: La posición de la bola no se conoce.

  Ejecutar: El robot gira 85 grados a la izquierda o a la derecha según un valor aleatorio.

- Buscar_Portería:

  Preparado: La posición de la portería contraria no se conoce.

  Ejecutar: El robot gira 85 grados a la izquierda o a la derecha según un valor aleatorio. En principio si un jugador conoce su posición absoluta en el campo, saber dónde están las porterías es inmediato.

- Ir_Bola:

  Preparado: La posición de la bola se conoce y la distancia a la bola es mayor de 2 metros.

  Ejecutar: El robot gira y avanza la distancia necesaria para acercarse a la bola. El cálculo se realiza con un algoritmo basado en cálculos matemáticos.

- Chutar_Gol

  Preparado: La distancia a la bola es menor de 2 metros (es chutable) y la dirección hacia la portería contraria es conocida. La decisión sobre si chutar a puerta o no se toma en la habilidad.

  Ejecutar: Chuta a la máxima velocidad (100) hacia la portería contraria. El algoritmo que implementa esta habilidad tiene en cuenta la posición de los contrarios a la hora de decidir cómo chutar. Además, comprueba si el disparo a puerta no se puede realizar por estar la bola en la posición opuesta a la portería. Si es así, realiza un auto-pase hacia un lado y luego chuta.

- Ir_Posición:

  Preparado: El jugador conoce su propia posición y ésta no es la que tiene asignada inicialmente. La decisión de si el jugador está en su posición o no, está definida permitiendo un grado de tolerancia que evite movimientos innecesarios.

  Ejecutar: Realiza los giros y avances apropiados para llegar a su posición, para lo que utiliza un algoritmo similar al de Ir_Bola.

- Pasar:

  Preparado: La distancia a la bola es menor de 2 metros y la posición de algún compañero es conocida. En el equipo implementado se incorporaron diversas versiones de pase, que permitían definir actitudes defensivas y ofensivas.

  Ejecutar: Chuta hacia la posición del compañero utilizando la fuerza calculada para que la bola le quede lo más cerca posible evitando, a la vez, que el pase pueda ser interceptado por el contrario.

- Mirar_Bola_Posición:

  Preparado: El jugador está en la posición del campo que tiene asignada y no está mirando de frente a la bola.

  Ejecutar: Girarse hacia la bola. Este es, por ejemplo, un comportamiento habitual del portero, consistente en estar en el centro de la portería mirando continuamente a la bola.

- Salir:

  Preparado: La posición de la bola se conoce y la distancia a ella es menor de 15 metros.

  Ejecutar: Gira y avanza hacia la bola. El hecho de que la bola se mueva se tiene en cuenta a la hora de decidir los movimientos del robot.

- Despejar:

  Preparado: La distancia a la bola es menor de 2 metros.

  Ejecutar: Chuta hacia la portería contraria tratando de evitar a los jugadores del otro equipo.

- Buscar_Compañero:

  Preparado: Siempre se puede ejecutar.

  Ejecutar: Gira 90 grados a la izquierda o derecha según un valor algoritmo.

- Avanzar_Puerta:

  Preparado: La distancia a la bola es menor de 2 metros.

Ejecutar: Chuta la bola y avanza hacia ella, es decir, intenta avanzar con la bola. En principio, el avance más utilizado es directamente hacia la portería contraria, pero existen otras posibilidades como paralelo a la banda, en horizontal o en diagonal. Todas ellas se implementaron como habilidades diferentes, pero realmente equivalen a ésta.

- Deambular:

  Preparado: No está a más de 7 metros de su posición asignada durante el juego.

  Ejecutar: Realiza un giro aleatorio y avanza también una distancia aleatoria entre 10 y 40.

- Ganar_Partido:

  Preparado: El tiempo ha finalizado.

  Ejecutar: El jugador muestra el resultado del partido y termina su ejecución.

Las habilidades empleadas en la RoboCup se diseñaron de forma heurística. Cada una de ellas se implementó como una clase de C++ derivada de `Habilidad_Fútbol`, derivada a su vez de la clase `Habilidad` descrita en la Formalización 4.1. Esta clase, como muestra la Formalización 6.2, incluye cuatro métodos que implementan las acciones contempladas en el simulador, así como el atributo para acceder a la información almacenada por el agente (atributo `Información`) y el atributo para enviar su resultado al simulador (atributo `Puerto`). Ambos atributos pueden considerarse una abstracción de los sensores y actuadores de un robot real.

De esta forma, la definición de una habilidad particular, por ejemplo la habilidad `Buscar_Bola` citada anteriormente, consiste simplemente en darle un nombre, implementando el método `di_nombre` y establecer sus métodos `preparado` y `ejecutar`.

La Formalización 6.3 muestra una versión simplificada en C++ de la habilidad `Buscar_Bola` según se acaba de definir. En ella se indica cómo se puede hacer uso de la información almacenada y cómo enviar las acciones al simulador.

En el primer caso utiliza el atributo `Informacion` para acceder a la información referente a la bola, utilizando el atributo `di_Bola`. Para comprobar si la información de la bola es válida o no, se comprueba si el tiempo en que se actualizó dicha información por última vez, más un índice de tolerancia (`Validez`), es mayor que el tiempo actual. Si lo es quiere decir que la información sobre la bola es reciente y por tanto no hay por qué ejecutar esta habilidad.

Formalización 6.2 Definición de la clase *Habilidad_Futbol*

```
class Habilidad_Futbol: public Habilidad{
protected:
  Sensores* puerto;
  Informacion* informacion;

public:
  Habilidad_Futbol(Informacion *, Sensores * );
  void Decir(char *);
  void Chutar(float, float);
  void Girar(float);
  void Avanzar(float);
};
```

En cuanto al envío de comandos al simulador, la acción utiliza uno de los métodos de la clase padre, `Girar`. Dicho método envía el comando `turn` al simulador. El valor del giro, como se indicó anteriormente, es 85 grados bien hacia un lado o hacia otro en función de un valor aleatorio.

Por supuesto, ésta es una de las habilidades más simples que se pueden construir, pero da una idea de las ventajas que aporta la orientación a objetos para ocultar la complejidad de la gestión de las comunicaciones, el mecanismo de interpretación de los mensajes, la actualización de las informaciones, etc.

Formalización 6.3 Definición de la clase *Buscar_Bola*

```
class Buscar_Bola: public Habilidad_Futbol{

public:
  void ejecutar() {
      Girar(Aleatorio_Giro( ) * 85.0);  }
  int preparado() {
      return (!(informacion->DiBola()->DiTiempo() + Validez >
                informacion->DiTiempoGlobal()) );  }
  char* di_nombre() {
      return("Buscar_Bola\0"); }
};
```

6.2.2 La Información Conocida por los Jugadores

Cada agente definido en ABC^2 puede utilizar las informaciones que almacene para la toma de decisiones. La cantidad de información y la gestión de la misma son competencia de cada agente en particular y no está definido explícitamente en el modelo ABC^2. Sólo se especifica que dicha información se guardará de forma estructurada y no estructurada.

En el caso de la RoboCup, el conjunto de información estructurada implica la generación de una serie de clases para almacenar las posiciones relativas de objetos, jugadores y de la bola. Esta clase no sólo almacena los valores, sino que proporciona métodos para acceder a ellos y modificarlos ordenadamente.

Además, la clase que almacena la información de una posición se encarga de mantener la información sobre el momento en el que se obtuvo dicha información de los sensores. Las habilidades utilizan esa información para asignar el grado de fiabilidad que a cada una le convenga según su cometido. En una primera versión, se realizó un sistema de estimación de las posiciones de los objetos no vistos, similar al propuesto en [Bowling *et al.*, 1996]. Sin embargo, este mecanismo demostró ser poco fiable, provocando más errores de los que solucionaba. Ello es debido a que es muy difícil hacer predicciones en un entorno tan dinámico como el de la RoboCup, donde además todas las informaciones recibidas son relativamente inciertas.

Se puede resumir la información estructurada que se guarda en forma de objetos de la clase `Posición_Relativa`, en la `Información` de un agente de la RoboCup en la siguiente lista:

- Posiciones relativas de las marcas del campo.
- Posiciones relativas de los compañeros.
- Posiciones relativas de los contrarios.
- Posición relativa de la bola.
- Posiciones relativas de las porterías.
- Posición absoluta del jugador en el campo

Así mismo, es interesante que el agente conozca ciertos aspectos aislados, no estructurados, como la situación del juego, el resultado, etc. Estas informaciones se

almacenan dentro de `Información` como atributos con un rango en general limitado de posibles valores. Los más importantes son:

- Número de marcas conocidas (18 en la versión actual).
- Clave de comunicaciones, para cifrar la información.
- Situación del jugador respecto al campo, si está dentro o fuera del terreno de juego.
- Situación del jugador respecto a su posición asignada.
- Tolerancia para aceptar que se encuentra en una posición.
- Lado del campo en que juega (izquierda o derecha).
- Resultado, goles del contrario y propios.
- Nombre del equipo.
- Número del jugador.
- Tiempo transcurrido.
- Fichero de traza.
- Activación del fichero de traza.

6.2.3 Definición del equipo

Una vez que se han definido las habilidades se pueden diseñar los jugadores. Un agente consiste, como se definió en la Sección 3.3, en la definición de sus habilidades, las relaciones entre ellas y la definición de sus heurísticas. Además, en el caso de la RoboCup, es necesario especificar la inicialización de cada uno de los jugadores. Para realizar dicha definición se genera un fichero de configuración para cada uno de los agentes.

La definición del equipo consiste simplemente en la enumeración de los jugadores que lo forman. Como se ha indicado, cada jugador está controlado por una instancia del software que implementa el modelo ABC^2, adaptado para la RoboCup. Por lo tanto, la definición del equipo consiste en un simple *script* en el que se especifica el

Formalización 6.4 Definición de un equipo para la RoboCup

```
#! /bin/bash
agent uc3m portero.dat &
sleep 1
agent uc3m defensa-izqd.dat &
sleep 1
agent uc3m defensa-cent.dat &
sleep 1
agent uc3m defensa-drch.dat &
sleep 1
agent uc3m central-izqd.dat &
...
```

nombre del ejecutable y el fichero de configuración que describe a cada jugador, como muestra la Formalización 6.4.

Cada uno de los ficheros de definición de un jugador consta de los siguientes apartados:

1. Parámetros iniciales.
2. Declaración de habilidades con su lista de necesidades.
3. Habilidad inicial.
4. Información sobre otros agentes (páginas amarillas).
5. Definición de las heurísticas.

Para mostrar cuál es la estructura de esta definición se presenta la definición de un portero simple. Su comportamiento debe consistir en situarse en el centro de su portería, permanecer en ese lugar siempre mirando a la bola y tratar de interceptarla si se aproxima a una distancia menor de 15 metros. Si consigue interceptarla, su acción inmediata debe ser tratar de despejarla chutando hacia la portería contraria, puesto que en el simulador no estaba prevista la posibilidad de coger la bola con la mano. Su definición se muestra en la Formalización 6.5.

Las líneas que comienzan por $*$ son comentarios y además sirven para separar las distintas partes. La primera parte fija la posición inicial en el campo, en coordenadas cartesianas del campo, correspondiendo la posición $(X = -50, Y = 0)$ al centro de su propia portería; la orientación, es decir, la dirección en la que mira (0 significa mirar

Formalización 6.5 Definición de un jugador (portero)

```
* Parámetros iniciales
-50 0 0 1 portero
* Habilidades
Ir_Posicion 0.7 { }
Buscar_Bola 0.6 { }
Mirar_Bola  0.75 { Ir_Posición Buscar_Bola }
Salir 0.8 { Mirar_Bola }
Despejar 0.9 { Salir }
Ganar 1 { Despejar }
* Habilidad Inicial
Ganar
* Habilidades de sus compañeros
Defensa_Izquierdo: Chutar, Pasar, Recibir, ...
...
* Definición Heurísticas
portero.heuristicas
* Fin_Fichero
```

de frente); y la tolerancia con la que acepta que la posición calculada corresponde con la fijada (1 indica que acepta un error de 1m en el cálculo de las posiciones). También se establece el nombre del agente (`portero`).

A continuación se fijan las habilidades que tendrá el robot. Para cada una de ellas se fija una *Prioridad* o peso, así como la lista de necesidades. Por ejemplo, la habilidad `Mirar_Bola` tiene dos necesidades, `Ir_Posición` y `Buscar_Bola`, y una prioridad inicial de 0.75.

Para que el sistema funcione correctamente es necesario que cada necesidad que aparezca en una habilidad haya sido previamente definida y tenga su implementación apropiada, proporcionando los métodos *Preparado* y *Ejecutar*.

Una vez que las habilidades del robot se han definido, es el turno de sus heurísticas. En la versión utilizada de ABC^2 se utilizaron reglas borrosas para implementar las heurísticas, con la misma estructura que las empleadas en el capítulo anterior. Sus variables de entrada representarán la situación y las de salida el nuevo peso de los actos. Para definirlas hay que especificar tres partes: las etiquetas para las entradas, las etiquetas para las salidas y las reglas propiamente dichas. Todo ello se realiza en un fichero específico, denominado en el ejemplo `portero.heurística`.

Para ello, y como muestra la Formalización 6.6, se define en primer lugar cada una de las entradas dando su nombre, por ejemplo `Distancia_Bola`. A continuación se es-

pecifican cada una de las etiquetas, por ejemplo *Muy_Cerca, Cerca, Lejos* y *Muy_Lejos.* La descripción de las etiquetas [Zadeh, 1973] se ha realizando enumerando los vértices de un trapecio de altura normalizada a uno definido sobre el dominio de las variables. Por ejemplo, *Muy_Cerca* se define como -1, 0, 2, 2.1. Es decir, se define *Muy_Cerca* entre los valores 0 y 2.1 metros. La definición a partir de -1, que a primera vista parece extraña, pues si es una distancia su rango debería comenzar en cero, se debe a razones de implementación de las reglas borrosas, que impiden diseñar el trapecio con alguno de sus lados completamente perpendicular al eje de abscisas. Por tanto, si se desea que el trapecio tenga la altura máxima en el origen (cero) el trapecio debe comenzar en algún punto negativo.

Formalización 6.6 Definición de etiquetas y reglas heurísticas del portero

```
* Entradas
{  Distancia_Bola
Muy_Cerca -1 0 2 2.1
Cerca      0 2 10 15
Lejos      8 15 20 25
Muy_Lejos 22 30 120 130 }
...
* Salidas
{ Salir
Disminuir -2 -1.2 -0.8 0
Igual -0.5 -0.2 0.2 0.5
Aumentar 2 1.2 0.8 0 }
...
* Reglas
{
Distancia_Bola es Muy_Lejos => Salir es Disminuir
Distancia_Bola es Cerca & Distancia_Posicion es..=> Salir es Aumentar &...
... }
```

A continuación se definen las salidas de las reglas del mismo modo. En este caso las variables representan los pesos de las distintas habilidades del robot. En la Formalización 6.6 se muestran las de *Salir.* Las etiquetas representan los grados de modificación del peso de esa habilidad. Por ejemplo, para *Salir* se han definido las etiquetas *Disminuir, Igual* y *Aumentar.* Estas etiquetas se definen de la misma forma que las etiquetas de entrada.

Por último, se definen las propias reglas. La definición se basa en reglas de primer orden compuestas por una condición, que utiliza el operador difuso AND, escrito

como &, la implicación escrita como => y el resultado, formado a su vez por clausulas separadas mediante *AND*.

Tanto la definición de las etiquetas como el diseño de las reglas se ha realizado de forma heurística, atendiendo a la experiencia conseguida en diversos partidos previos. En cualquier caso, eso no impide que se puedan emplear métodos de aprendizaje automático para generarlas o mejorarlas, puesto que están definidas con un lenguaje de alto nivel. Un ejemplo más detallado de uso de ABC^2 en este dominio se presenta en el Anexo B.

6.3 Resultados en la RoboCup

Una implementación del modelo descrito se probó en la competición de simulador de la RoboCup'97, en Nagoya (Japón). El equipo perdió (1-9) su primer partido contra CMUnited, uno de los dos equipos presentados por la Carnegie Mellon University. Ganó frente al RMKnights (10-0) y perdió frente a uno de los dos equipos presentados por la Kinki Universtiy (0-8). En resumen, dos partidos perdidos, uno ganado, 17 goles en contra y 11 a favor.

Antes de comentar los partidos individualmente, hay una primera consideración que conviene realizar referida a los marcadores. Se detectó en la competición, y se comentó en el *Workshop* correspondiente, que si un equipo era superior a otro en algún aspecto, el resultado tendía a ser muy elevado. Hubo resultados del orden de los treinta goles de diferencia. Esto es debido a que la competición se celebra en un simulador, lo cual resalta las diferencias. La mayoría de las veces una sola de las características del equipo (buena o mala) eclipsaba la mayoría de las otras. En consecuencia, el análisis del rendimiento de un equipo debe intentar encontrar el elemento que hace que un equipo sea mejor que otro, más que el resultado numérico en sí mismo.

Para realizar el análisis de los partidos celebrados se va a utilizar una representación como la de la Figura 6.4 que resume el partido. Estas figuras representan el desarrollo del partido, indicando en qué campo se encontraba la pelota, los disparos a puerta y los goles.

El primero de los gráficos, Figura 6.4, muestra el partido contra el equipo de la Carnegie Mellon. Este equipo es uno de los fundadores de la RoboCup y es un equipo realmente muy refinado. El grupo de la profesora Manuela Veloso ha estado trabajando en este dominio durante bastante tiempo [Stone and Veloso, 1995], y ha conseguido

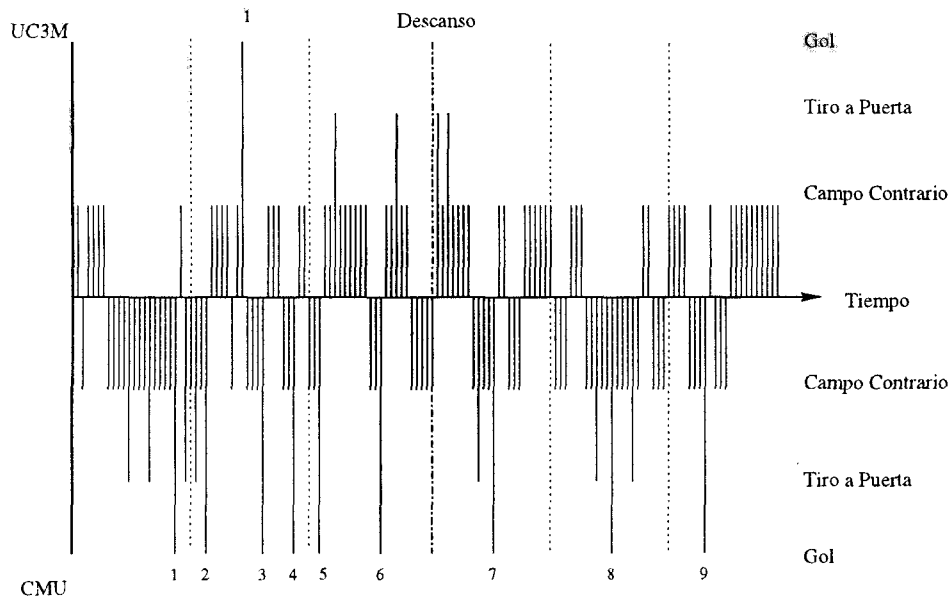

Figura 6.4: Partido Carlos III contra CMUnited (1-9).

unos jugadores con unos comportamientos individuales notables y una estrategia muy acertada. En resumen, se trataba de uno de los mejores equipos que compitieron, tanto por su historial como por el resultado que consiguieron.

La única objeción a este equipo, desde el punto de vista de la RoboCup, es que no corresponde a un modelo genérico de control de robots como puede ser ABC^2. Se trata de un software muy especializado, diseñado fundamentalmente por Peter Stone [Stone and Veloso, 1997b]. Es decir, conocían muy bien el entorno y han diseñado un programa que juega bien en ese entorno. Por ejemplo, el rendimiento de sus jugadores fue realmente bueno, dispararon a puerta en 16 ocasiones y consiguieron 9 goles, una efectividad del 56 por ciento.

Los jugadores de nuestro equipo dispararon a puerta sólo en 4 ocasiones y consiguieron tan solo un gol, un 25 por ciento de efectividad, lo cual indica que o nuestro portero era muy malo o sus delanteros muy buenos. Analizando cualitativamente el partido queda claro que es lo primero: algunas de las habilidades tienen que ser claramente mejoradas, en especial la del portero.

El segundo partido fue contra el equipo RMKnights, Figura 6.5. El equipo de Simon Ch'ng estaba probando también el modelo que éste desarrolla en su tesis doctoral en el banco de pruebas de la RoboCup, como comentamos durante la misma. El principal defecto de su equipo resultó ser que sus jugadores eran demasiado lentos. Esto hacía que los jugadores controlados mediante ABC^2 fuesen capaces de reaccionar antes que

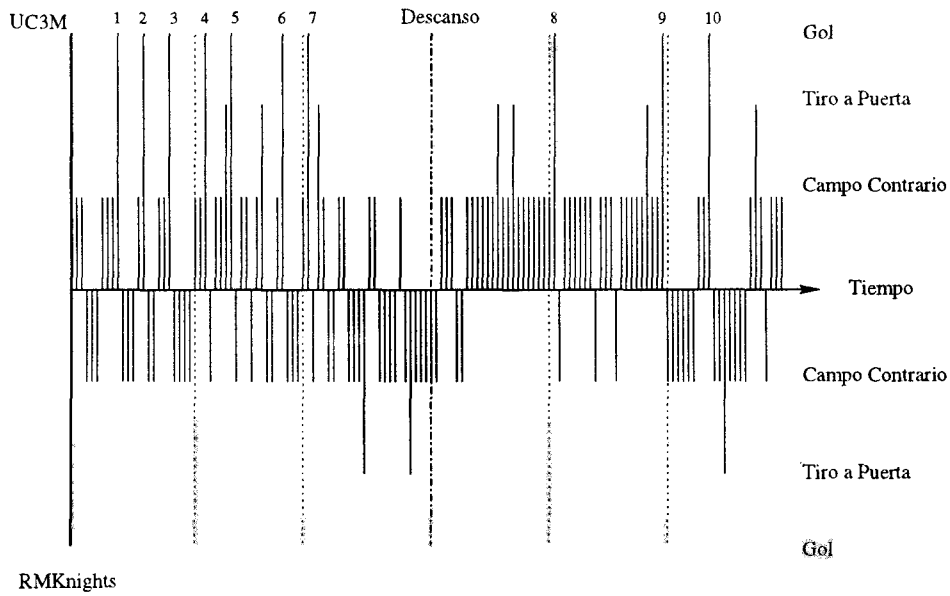

Figura 6.5: Partido Carlos III contra RMKnights (10-0).

los suyos, llegando a coger la bola antes, incluso estando inicialmente más lejos de ella. Así pudimos comprobar que ABC^2 realmente funciona bien en entornos altamente dinámicos.

El tercer partido, Figura 6.6, correspondió al encuentro con el equipo Niken de la Universidad de Kinki (Japón). En nuestra opinión es otro claro ejemplo de cómo el entorno refuerza el efecto de una característica particular. Así, consideramos que una de las razones fundamentales de la derrota fue el control del parámetro denominado "stamina" de los jugadores. Este parámetro indica cómo de "cansados" están los jugadores. Dicho parámetro no está accesible para los jugadores, se maneja de forma interna desde el simulador, y su efecto principal es que los jugadores que se encuentran "agotados" corren menos y chutan con menos fuerza.

La estimación del parámetro "stamina" por parte del equipo Niken pareció funcionar. Así, sus jugadores podían moverse más rápido que los nuestros en determinados periodos del encuentro. Esto tiene su reflejo en los momentos en los que consiguieron la mayoría de sus goles. Casi todos fueron obtenidos en la última parte de cada mitad.

En resumen, se trataba de probar simplemente si ABC^2 se podía emplear en este dominio, y se ha visto que sí. En cuanto a las habilidades empleadas, claramente muchas de ellas deben ser ajustadas. En especial el comportamiento del portero.

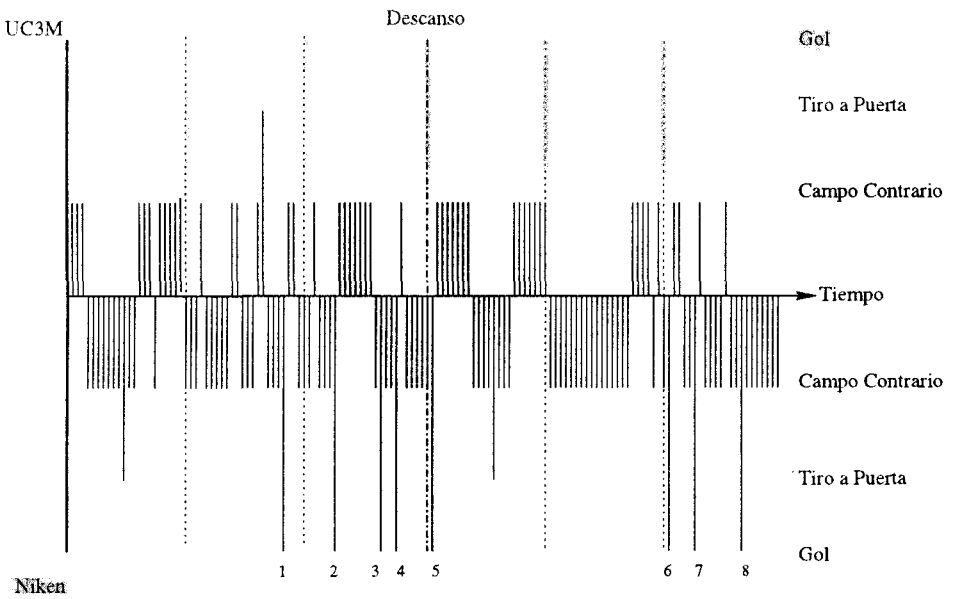

Figura 6.6: Partido Carlos III contra Niken (0-8).

6.3.1 Comparación con las estrategias de otros equipos

Además de analizar el rendimiento del equipo, la RoboCup sirvió para discutir con los demás competidores sus estrategias, su visión del problema etc., que es el objetivo de la RoboCup, más que la propia competición.

El equipo más documentado [Stone and Veloso, 1997a, Stone and Veloso, 1997b, Stone and Veloso, 1998, Bowling *et al.*, 1996] es sin duda el CMUnited, uno de los más antiguos. Además, el hecho de haber compartido algunas semanas con ellos en el verano de 1997, cuando preparaban su equipo, le hace aún más conocido para el autor de esta tesis. La estructura del equipo se basa en definir posiciones fijas para los jugadores, lo que denominan una "decisión de vestuario". Así, tiene posibles configuraciones del equipo 4-3-3, 4-4-2, etc. Dichas configuraciones están "cableadas" en el software (lo cual le lleva a recompilar el software cuando decide cambiarla) de los agentes. El software en sí es el mismo, el comportamiento de cada agente viene dado por el número de jugador que le corresponde.

Una vez situados los jugadores, estos tienen que tomar sus decisiones. Para ello distingue entre sus acciones básicas: interceptar la bola, ir hacia ella, etc., que se implementan en general como algoritmos matemáticos (también realizaron distintos experimentos utilizando una red neuronal [Stone and Veloso, 1997a]). Para ello emplean un árbol de decisión basado en factores de confianza [Stone and Veloso, 1997b] en el que tienen en cuenta factores como el entorno del agente, el entorno del receptor

etc.

En ese sentido, se puede considerar que CMUnited es una versión evolucionada del equipo ganador de la Pre-RoboCup, Ogalets'96. Este equipo, utilizado como "sparring" por la mayoría de los participantes, juega considerando que los jugadores están en posiciones fijas, por lo que un jugador que consiga la bola simplemente intenta pasarla allí, esté o no esté el compañero, lo que le permitía gran velocidad.

El fundamento de RMKnights [Ch'ng and Padgham, 1997] es algo diferente, ya que propone un modelo basado en la definición de los papeles *roles* de los agentes. Cada uno de esos papeles conlleva una serie de "responsabilidades", es decir, metas que implica su *rol*. Para conseguir dichos roles dispone de una serie de "estrategias", cada una de las cuales está disponible bajo ciertas condiciones y está compuesta por una serie de acciones que el agente debe realizar. Dichas acciones pueden implicar a su vez que otros agentes tomen otros *roles*. El control del agente en este modelo pasa por decidir, mediante una serie de heurísticas, el *rol* que el agente tomará en un momento determinado. Por último, mantiene un esquema de reactividad, independiente de todo el sistema anterior, para responder en los casos en que las acciones previstas por la estrategia dejen de ser posibles.

Un modelo similar a este último, también basado en roles, es el de Silvia Coradeschi La primera aplicación de su modelo [Coradeschi *et al.*, 1996] fue en simuladores de combate aéreo y después en el entorno de la RoboCup [Coradeschi and Lars, 1997]. En este modelo el concepto de rol se considera fundamental; los agentes pueden actuar y coordinarse con otros agentes dependiendo de su rol. El mecanismo de decisión del agente se especifica en forma de reglas priorizadas organizadas en forma de árbol, donde se tienen presentes tanto la reactividad, como la capacidad de perseguir metas de alto nivel. La coordinación entre los agentes se realiza fundamentalmente mediante tácticas estratégicas y observaciones de acciones de cada agente (distinguido por el número en el caso de la RoboCup), más que mediante comunicación explícita.

Por último, un sistema interesante aunque todavía en desarrollo para su aplicación en la RoboCup, fue el presentado por el grupo de Edmund H. Durfee. Está basado en un modelo general de razonamiento denominado UM-PRS, Sistema de Razonamiento Procedimental (PRS) de la Universidad de Michigan (UM), basado a su vez en el sistema PRS desarrollado por Georgeff [Georgeff and Lansky, 1986]. UM-PRS es un sistema que se desarrolló originalmente para la realización de tareas militares coordinadas entre distintos vehículos, por ejemplo, el reconocimiento de zonas, avances, etc.

Para ello, se parte del concepto de "plan de la misión" donde se especifican una serie de metas, para conseguirlas dispone de una base datos con planes (procedimientos para conseguirlas) que un modulo de razonamiento elige en función de la información del entorno. El hecho de utilizar UM-PRS para implementarlo facilita, por ejemplo, el reconocimiento de los planes del oponente, ya que UM-PRS es capaz de generar una red bayesiana para predecir sus acciones a partir de sus observaciones y suposiciones sobre el oponente.

6.4 Fútbol con robots reales

La RoboCup no es la única competición de robots basada en el fútbol. Concretamente se consideró la utilización de ABC^2 para competir en el torneo de fútbol entre mini-robots Khepera organizado en paralelo con la *European Conference on Artificial Life* (ECAL) de 1997 celebrada en Brighton (Reino Unido) del 28 al 31 de Julio de 1997.

La competición en este caso consistía en partidos uno contra uno entre mini-robots Khepera. Las normas básicas de la competición eran las siguientes:

- Los robots debían "vestir" una camiseta de 15 cm de alto consistente en un cilindro con bandas blancas y negras de una pulgada cada una.
- Los robots deberían llevar todo el hardware y software necesario *on-board*, es decir, no pueden recibir ayuda de ningún tipo desde el exterior.
- El campo constaba de un muro negro de 70 cm por 90 cm, en el que las porterías eran huecos blancos en el muro, de 30 cm de ancho y 20 de alto, justo en el centro de cada uno de los lados cortos del rectángulo que forma el campo de juego.
- La bola era una pelota de tenis amarilla.

Todas estas normas estaban pensadas para que los jugadores utilizasen una cámara, sacada al mercado por el patrocinador de la competición poco tiempo antes, para distinguir a los jugadores de la pelota y de la portería. El principal problema para no participar en esta competición fue precisamente el hecho de no disponer de dicha cámara. Aún así, se intentó el diseño de algunas habilidades con los sensores básicos del Khepera y combinándolos mediante ABC^2 a fin de probar su idoneidad en esta competición.

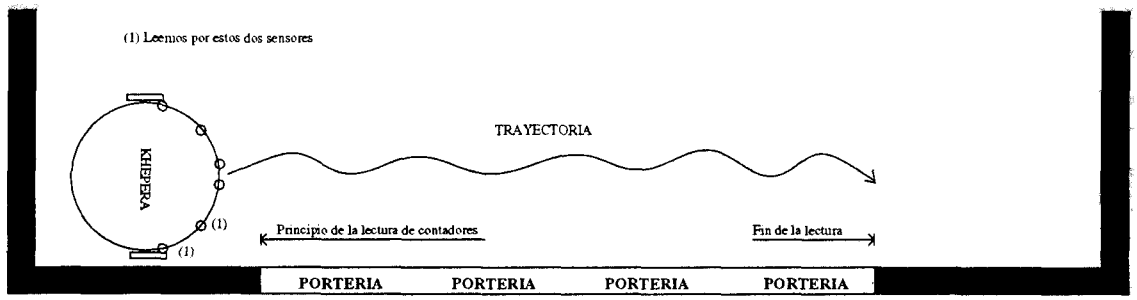

Figura 6.7: Comportamiento de un portero real

Como ejemplo, se analiza uno de estos comportamientos: el portero. Dada la poca utilidad de los sensores se decidió implementar un portero que se dedicase a recorrer sin parar el frente de la portería como muestra la Figura 6.7. Para ello se utilizan los sensores que forman 45 y 90 grados con la dirección del robot y miran hacia la portería (se supone que el portero parte en un extremo conocido de la portería). Si el valor medido por dichos sensores es menor que un cierto valor, el robot se tiene que acercar a la portería y si es menor alejarse.

El mayor problema es que las medidas devueltas por los sensores son de muy difícil interpretación. Se obtiene el mismo valor numérico al estar lejos de una superficie blanca (como la portería) que mucho más lejos de una negra (como las paredes). La solución es realizar medidas periódicas, interpretando el cambio brusco como el principio / final de la portería. Esto es lo que muestran las medidas de la Figura 6.7.

A todo ello hay que sumar los posibles errores. Por ejemplo, si por alguna razón el robot no queda perfectamente alineado después de girarse completamente, puede intentar corregir su posición de forma equivocada y acabar chocando con la pared o con la portería, como muestra la Figura 6.8. Por todo ello, una vez el diseño de la habilidad se ha realizado de forma experimental, basado en el principio del ensayo-error, se ha seguido depurando hasta que se ha conseguido un comportamiento razonable.

De igual forma se han implementado otras habilidades como, por ejemplo, *Evitar_Contrario*, lo cual implicaba realizar una serie de medidas para distinguirlo de la bola; *Chutar_Bola*, que implicaba en primer lugar reconocerla y luego tomar carrerilla y chocar con ella; *Empujar_Bola* que en vez de chocar bruscamente intentaba arrastrarla; *Recorrer*, que hacía que el jugador recorriera el campo en zig-zag, e *Ir_Posicion*, que es capaz de llevar al jugador desde cualquier lugar del campo a su posición inicial en el centro de la portería.

Finalmente indicar que ABC^2 es capaz de fusionar estas habilidades para conseguir

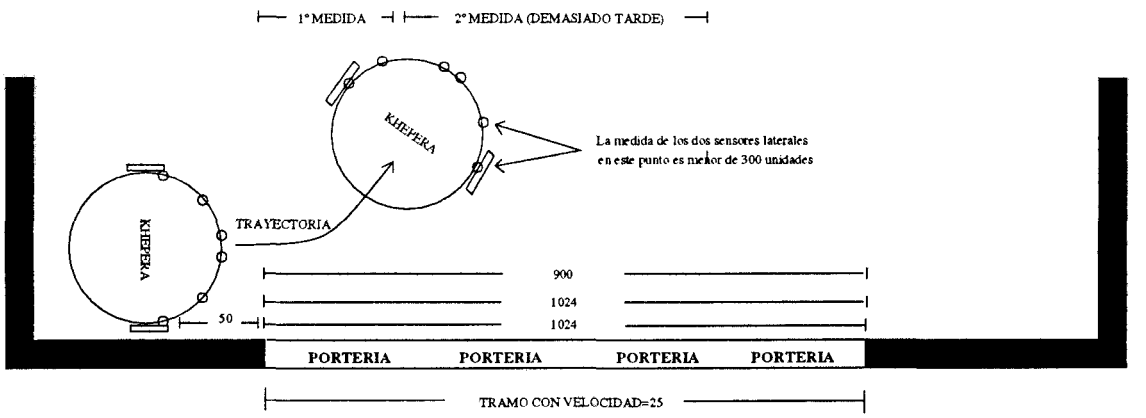

Figura 6.8: El portero pierde la orientación

un comportamiento más complejo como el mostrado en la Figura 6.9.

Esta figura muestra la evolución de un mini-robot khepera con los comportamientos descritos en el párrafo anterior. La idea es rematar a gol la pelota, donde se supone que hay que evitar a un jugador contrario. Así, en la primera imagen (superior izquierda) el jugador se encuentra con el contrario. Para reconocerlo realiza algunas medidas adicionales, que le implican realizar ciertos movimientos (segunda y tercera imagenes). Ese jugador se supone estático, porque el proceso de distinguir la bola del contrario solo con seis valores numéricos de proximidad requiere esas medidas adicionales.

Una vez decidido que el objeto encontrado es el jugador contrario se evita (cuarta imagen). A continuación se localiza la bola y se va hacia ella (quinta imagen), se la reconoce (sexta imagen), se toma carrerilla (séptima imagen) aplicando la habilidad *Chutar_Bola* y se golpea la bola (última imagen).

Para concluir, indicar que las competiciones entre robots no son algo nuevo. La AAAI (American Association for Artificial Intelligence) lleva varios celebrando una competición. El año 1997 esta competición pasó, además, a ser una competición de multi-robots [Collins and Balch, 1997]. Además, debido al éxito de la RoboCup están surgiendo otras competiciones basadas en el fútbol entre robots. Por ejemplo en Francia se celebrará también el año próximo el EuroBot98 [3], organizado por una cadena francesa de televisión. Así como el campeonato de la FIRA, similar a la RoboCup pero con robots más reducidos[4]. Incluso en España empiezan a surgir concursos como el promovido por los estudiantes del IEEE[5].

[3]http://anstj.mime.univ-paris8.fr/robotique/COUPE_E=M6/coupe.html
[4]http://www.fira.net/fira
[5]http://www.ieeesb.upm.es/~jjsan/

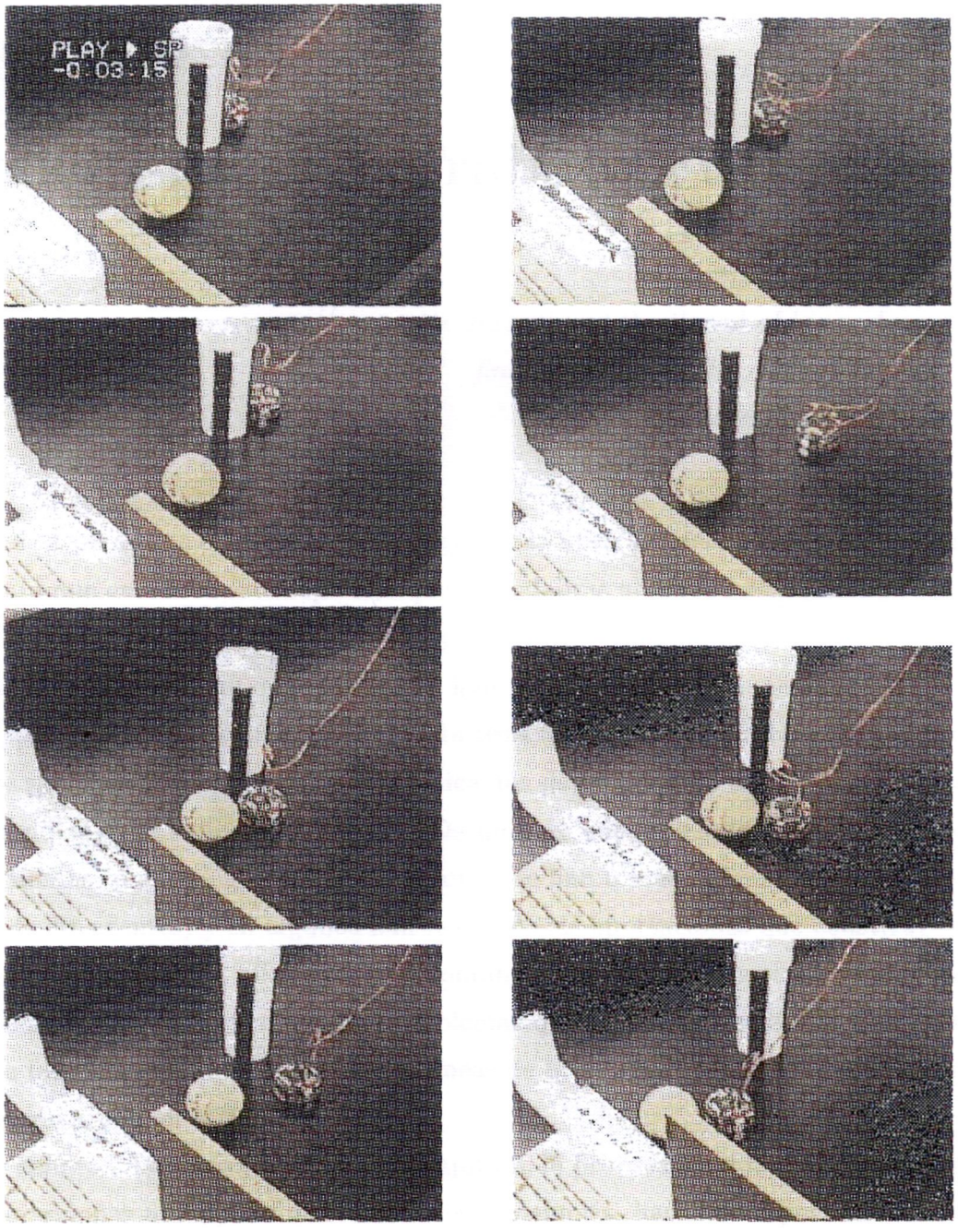

Figura 6.9: Khepera jugando al futbol.

Capítulo 7

Conclusiones y Trabajo Futuro

He draweth out the thread of his verbosity
finer than the staple of his argument.
William Shakespeare

Es de esperar que el lector no haya llegado a la misma conclusión que el genial autor de la cita que encabeza este capítulo: que las palabras no hayan ocultado las ideas que se trataban de exponer en los capítulos precedentes. En cualquier caso, este capítulo resume las ideas más importantes que se han aportado en esta memoria.

El abanico de temas abordados en esta tesis ha sido grande, incluyendo, por ejemplo, los sistemas de planificación, la robótica, los sistemas multi-agente, los mecanismos de aprendizaje, la lógica borrosa, etc. De unos se han utilizado sus resultados como herramientas (p.e. la lógica borrosa), de otros se han tomado ideas o experiencias (sistemas multi-agentes) y en algunos han realizado las aportaciones que constituyen esta tesis. Ahora bien, todos ellos tienen en común la cantidad de cuestiones que permanecen abiertas y el enorme número de problemas que siguen apareciendo. Este capítulo resume también cuáles de estas nuevas líneas de investigación podrían beneficiarse del trabajo realizado en ABC^2.

En cuanto a la organización de este capítulo, la primera sección se dedica a resumir las aportaciones principales descritas en esta memoria, haciendo especial énfasis en la integración entre reactividad y planificación. La segunda presenta aquellas cuestiones que han quedado abiertas y que pueden beneficiarse del trabajo realizado.

7.1 Aportaciones principales

El resultado tangible de esta tesis es un modelo adaptable y modular para el control de robots autónomos. Este modelo emplea comportamientos predefinidos e independientes de tipo reactivo, que es capaz de componer de forma oportunista para obtener un comportamiento inteligente y que pueda ser coordinado con otros agentes.

Este resultado puede descomponerse en una serie de contribuciones que pueden agruparse en varios temas. La organización de las mismas, en orden de relevancia, corresponde a los apartados siguientes.

Integración de Reactividad y Planificación

Una de las principales aportaciones del modelo desarrollado es la capacidad de integrar las ventajas de los sistemas reactivos, en cuanto a velocidad, adaptabilidad, etc.; con la versatilidad de los sistemas simbólicos. Para ello parte de un planteamiento distinto al de estos últimos: no utiliza el concepto tradicional de "estado" para realizar un razonamiento hipotético sobre la evolución del mismo. Tampoco realiza ningún razonamiento sobre la estructura de información que mantiene (modelo del mundo) con el fin de predecir estados posibles. El mecanismo de decisión de las acciones utiliza la situación real del mundo y las oportunidades de actuación que se generan en cada momento.

En cierto modo, se puede considerar similar a los modelos reactivos. Sin embargo, los sistemas reactivos sólo generan una acción, que se considera la más apropiada, en cada instante del tiempo. En estos sistemas esa única acción es la que se ejecuta. ABC^2 contempla la posibilidad de que distintas acciones sean ejecutables en un determinado instante, proporcionando un mecanismo, para decidir cuál de las posibles acciones se ejecutan.

Además, las listas de necesidades que tiene cada acción es otro mecanismo que permite encadenar una acción con aquéllas que la hacen posible. En ese sentido, se pueden considerar los árboles resultantes de encadenar las listas de necesidades como "planes precompilados".

Esto hace que el sistema no se base en el principio reactivo de que las acciones pasadas no influyen en las presentes. Este modelo tiene cierta memoria, que se fundamenta en la forma en que se han insertado los actos en la agenda. Concretamente, en el atributo que marca el instante en que se insertaron, utilizado por las heurísticas como un factor más a la hora de decidir la siguiente acción. Es decir, las condiciones

particulares del entorno influyen, pero no deciden, las acciones del agente.

En resumen, empleando ABC^2 un agente puede cambiar sus metas sin cambiar la forma en que sus acciones se disparan. Además, se puede especificar la forma en que se encadenan las acciones. O lo que es lo mismo, tiene capacidad de planificación "a priori" con ejecución reactiva.

Tratamiento de las Contingencias

Se pueden definir las contingencias como eventos no previstos. Durante mucho tiempo han sido la fuente de mayor crítica a los sistemas de planificación tradicional, como por ejemplo los efectos colaterales no previstos de las acciones, las imprecisiones en las informaciones, las acciones fallidas, etc. En general, se da por supuesto que los sistemas reactivos tratan más eficientemente este tipo de contingencias.

En ese sentido, ABC^2 aprovecha igualmente el carácter reactivo de la definición de las habilidades. Por otra parte, existe un tipo particular de contingencias, las que se producen en los entornos multi-agente, que los sistemas reactivos son incapaces de tratar adecuadamente. Estos sistemas carecen de representación simbólica para las acciones (operadores en planificadores simbólicos), con lo que difícilmente pueden solicitárselas unos a otros. Todo lo más son capaces de "reaccionar" a las acciones de los otros agentes. Lo mismo ocurre con la posibilidad de compartir información: la ausencia de un modelo les impide hacerlo.

Para los sistemas tradicionales, basados en la búsqueda en espacios de estados, significa un nuevo factor de crecimiento en las posibilidades, como cualquier otra contingencia. Por ello, tampoco parecen ser un mecanismo fiable.

El diseño del modelo expuesto, y su implementación, tienen en cuenta la posible existencia de otros agentes y aportan un mecanismo para tratar las contingencias que implica. Los otros agentes pueden, por ejemplo, realizar peticiones en cualquier momento, enviar datos, etc.; es decir, generan eventos asíncronos y no previsibles.

Por lo tanto, el modelo presentado permite tratar las contingencias del entorno de forma reactiva y además permite integrar en el mismo acciones decididas por el agente o pedidas por otros.

Aprendizaje de Comportamientos

Una aportación colateral en el campo del aprendizaje se ha presentado en el Capítulo 5. Consiste en un método para la adaptación de controladores borrosos basado en los

algoritmos genéticos.

El objetivo del experimento era mostrar que el modelo no tiene problemas para integrar comportamientos diseñados por un programador, o aprendidos por métodos artificiales. Pero, además, el método de aprendizaje desarrollado para probarlo constituye en sí mismo una aportación al aprendizaje subsimbólico [Matellán *et al.*, 1995a].

El mecanismo de aprendizaje desarrollado se empleó tan solo en el aprendizaje de las reglas. Leves modificaciones del mismo permiten su extensión al aprendizaje de las etiquetas borrosas, o conceptos. Lo cual lo hace interesante como mecanismo de emergencia de conceptos simbólicos a partir de mecanismos subsimbólicos.

El procedimiento descrito puede aplicarse también al aprendizaje de las reglas heurísticas del modelo, o en general, a cualquier sistema de razonamiento borroso.

Otras Aportaciones

La modularidad del sistema permite integrar acciones implementadas mediante distintos paradigmas de la inteligencia artificial como las redes neuronales, los controladores borrosos, sistemas estadísticos y matemáticos, etc.

El trabajo realizado para construir y probar el modelo realizado ha conllevado además diversas actividades, realizadas en colaboración con los compañeros del LAI, que merecen al menos una reseña. Por ejemplo, el desarrollo del simulador distribuido SimDAI [Sommaruga *et al.*, 1996], que va ya camino de su tercera versión, con objetos móviles y rediseño incluido; el desarrollo de un mecanismo de integración de comportamientos reactivos [Matellán *et al.*, 1995b] para el control de robots; el empleo de técnicas borrosas en diversos dominios [Molina *et al.*, 1996] o el aprendizaje genético expuesto anteriormente.

Por último, se ha demostrado que la integración entre experimentos simulados y los realizados con robots reales no tiene por qué ser traumática. Es más, si el simulador trata de imitar realmente el mundo real, modelando los sensores y las imperfecciones del mundo, el paso de un dominio a otro puede resultar sencillo. Visto de otra forma, el modelo es válido tanto para sistemas simulados como reales, como prueba el experimento del Capítulo 5.

7.2 Desarrollos futuros

El modelo propuesto en esta tesis proporciona nuevas perspectivas de solución para algunos de los problemas tradicionales de la planificación y robótica inteligente, etc.

Esas posibilidades conllevan diversas líneas de trabajo que se pueden agrupar en tres tipos:

1. La profundización en los temas abordados.
2. La adaptación del modelo a diferentes tipos de dominios.
3. La integración del modelo con otros campos de investigación.

En el primer caso, las futuras líneas de trabajo que se vislumbran se podrían resumir en:

- Generalización del uso de mecanismos de aprendizaje automático, simbólico o subsimbólico, para la generación de habilidades o su adecuación a las condiciones del dominio.
- Generación dinámica de las listas de necesidades. Es decir, emplear planificadores eficientes que permitan modificar la definición de las habilidades durante la ejecución..
- Aplicación también de las técnicas de aprendizaje automático para la generación y adaptación de las reglas heurísticas, empleando para ello reglas del tipo más apropiado.
- Profundización en el estudio de modelos de cooperación, integrando modelos basados por ejemplo en KQML [Finin *et al.*, 1994], o implementando mecanismos de asignación de tareas como la red de contratos.
- Realización de herramientas amigables para facilitar el diseño de las habilidades y las reglas heurísticas, añadiendo a dichas herramientas mecanismos formales que permitan detectar, por ejemplo, ciclos en las listas de necesidades, inconsistencias en las reglas heurísticas, etc.
- Implementación de mecanismos que permitan la modificación dinámica del software, permitiendo añadir por ejemplo nuevas reglas heurísticas, habilidades, etc. Lo cual puede permitir adaptar dinámicamente el agente a nuevas condiciones del entorno o a mejorar su rendimiento, mediante su integración con mecanismos de aprendizaje.

En cuanto a su adaptación a nuevos dominios:

- Incluisión del coste estimado de las habilidades en su definición y de las comunicaciones. De esta forma las heurísticas puedrían definirse también en función del coste de los actos.

- Adaptación del modelo desarrollado parece apropiado para abordar problemas típicos de tiempo real. Para ello sería necesario aplicar restricciones temporales a la ejecución de las habilidades e implementar razonamiento sobre el tiempo a las heurísticas.

- Integración del sistema de razonamiento con sistemas sensoriales más potentes, relacionados por ejemplo con la visión. En los experimentos realizados se ha utilizado un robot de reducida capacidad sensorial (8 sensores de infrarrojos) o un sistema preprocesado (el simulador de la RoboCup).

- Particularización de los métodos para aplicaciones concretas de carácter industrial.

En lo referente a la integración de ABC^2 con otros campos sería interesante estudiar las sinergias resultantes de:

- Emplear el modelo desarrollado para el control de agentes software. Por ejemplo, para realizar búsquedas en la red, comparación de precios en compras, etc.

- Desarrollar la idea de planificación oportunista para su uso en problemas tradicionales de planificación de tareas y búsqueda en espacios de estados.

- Utilizar los desarrollos en sistemas de reconocimiento de intenciones para mejorar la adaptabilidad del sistema y a la vez para probar la validez de los mismos.

- Estudiar la posible integración con sistemas operativos reducidos (micro-kernels) que den soporte directo, por ejemplo a nivel de *scheduler* para ABC^2.

Apéndice A

Lógica Borrosa

En distintos apartados de esta memoria se ha hecho uso de la lógica borrosa como herramienta, tanto para diseñar las habilidades de los robots en distintos dominios (bien diseñándolos heurísticamente, o bien, utilizando mecanismos de aprendizaje automático), como para establecer las reglas heurísticas que controlan el proceso de selección de los actos de la agenda de ABC^2. De esta forma se comprueba cómo es éste un paradigma que puede ser ampliamente utilizado en diversos dominios.

Sin embargo, en ninguno de los capítulos en los que aparece se ha hecho una introducción formal a la lógica borrosa ni al uso de la misma en el campo de la robótica cognitiva. Para paliar esta ausencia se resumen en este apéndice algunos de sus principios.

Este apéndice se divide en tres partes. En la primera sección se introduce brevemente el concepto de difuso o borroso y se describen brevemente las aplicaciones clásicas del mismo al campo de la robótica. En la segunda se realiza una descripción más detallada de los fundamentos de estos sistemas de razonamiento. Por último, la tercera se dedica a introducir el diseño de las reglas borrosas y los problemas que pueden surgir, describiendo el funcionamiento concreto de un controlador basado en lógica borrosa, especificando qué variables se consideran y cuál es el formato de las reglas.

Introducción a la lógica borrosa

Los controladores borrosos se han utilizado profusamente para el control de distintos tipos de robots autónomos. Por ejemplo, en [Reignier, 1994] se presenta la arquitectura LIFIA. Esta arquitectura es una jerarquía de niveles en la que cada uno de ellos trabaja de forma asíncrona, con su propio nivel de abstracción de datos, en la que el módulo de navegación consiste en un módulo reactivo basado en lógica borrosa. Otro

ejemplo clásico de sistema autónomo controlado mediante comportamientos borrosos es [Pin and Watanabe, 1993]. Este sistema consiste en un coche que se mueve de forma autónoma utilizando distintos conjuntos de reglas borrosos.

Otro ejemplo clásico es la arquitectura basada en lógica borrosa del robot Flakey [Saffiotti *et al.*, 1995]. Este robot integra capacidades de planificación de alto nivel, implementadas utilizando una abstracción denominada *behavior schemas*, que enlaza los movimientos físicos con las acciones abstractas usando operaciones de lógica multi-evaluada. De esta forma, metas y comportamientos reactivos se combinan para producir comportamientos más complejos.

El concepto de lógica borrosa se debe a Lofti A. Zadeh y tiene su origen en la necesidad de poder describir propiedades de forma vaga. Por ejemplo, dado un concepto como $Cerca$, la propiedad de que la "Distancia_A($Objeto_i$) es $Cerca$" tiene muchas más caracterizaciones que "Verdadero" o "Falso". En un primer paso se podría establecer una lógica multi-valuada, es decir, definir un conjunto mayor de posibles valores. Por ejemplo, establecer un rango como "Falso, Dudoso, Probable, Verdadero".

El siguiente paso sería establecer un rango continuo de valores. Esa es la aportación de la lógica borrosa, la definición del "conjunto borroso" y de las funciones de pertenencia, que se nombran como μ, de un concepto a un conjunto borroso. En general estas funciones se definen con valores normalizados continuos entre 0 y 1. De esta forma se podría definir el concepto $Cerca$ mediante una función de pertenencia borrosa $\mu_{Cerca}()$. Y a continuación, se podría decir que el grado de verdad de la expresión "Distancia_A($Objeto_i$) es $Cerca$" es $\mu_{Cerca}(Distancia_A(Objeto_i))$.

Controlador Borroso

En resumen, un subconjunto borroso A de un universo del discurso U viene dado por una función de pertenencia $\mu : U \rightarrow [0, 1]$ que relaciona cada elemento y de U, con un valor $\mu_A(y)$ que representa el grado de pertenencia de y a A. A esta operación se le denomina "borrosificación" (del inglés *fuzzyfication*), y depende de la implementación de cada aplicación. Su efecto es la transformación de un conjunto no borroso (la medida de un sensor) en un conjunto borroso. Esta operación corresponde a la parte izquierda de la Figura A.1, que consiste básicamente en usar la definición de las funciones de pertenencia (VN, N, F, VF), representadas en la figura como trapecios.

La principal ventaja de la teoría de los conjuntos borrosos es que permite trabajar

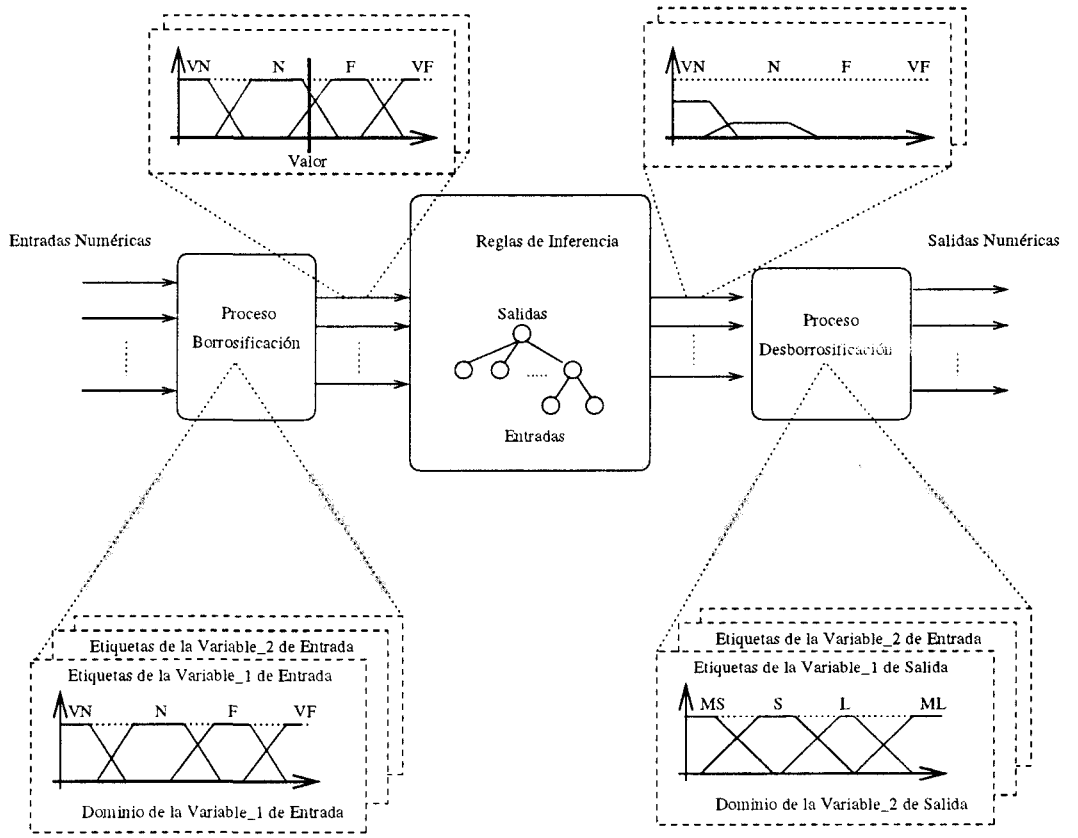

Figura A.1: Proceso de Razonamiento Borroso.

de forma lingüística con valores numéricos, sin perder la indeterminación intrínseca del lenguaje. Es decir, permite trabajar con variables cuyos valores son frases de un lenguaje natural o artificial; es decir, la concatenación de términos atómicos: etiquetas (adjetivos), modificadores (como "muy", "poco', "algo", etc.), operadores como la negación y usar marcas como los paréntesis.

Para ello la teoría de los conjuntos borrosos define el mismo tipo de operaciones que se realizan en la lógica tradicional. Estas operaciones están ahora basadas en las funciones de pertenencia. Para cada una de las operaciones existen varios métodos distintos que permiten calcular el conjunto borroso resultante.

Cada uno de esos métodos mantiene ciertas propiedades matemáticas como la asociatividad, distributividad, etc. entre las operaciones básicas de conjunción y disyunción. Las más conocidas son las parejas intersección-unión, mínimo-máximo, producto-producto y la de Luckasievicz. La Tabla A.1 muestra la definición de las tres últimas. La negación, **NOT X**, en general se implementa como $1 - \mu_X(x)$. Los modificadores lingüísticos (mucho, bastante, etc.) pueden implementarse de la misma forma.

Para la implicación, los métodos más habituales son la implicación de Luckasiewicz

	X **AND** Y	X **OR** Y
Min - Max	$\text{Min}\{\mu_X(x), \mu_Y(y)\}$	$\text{Max}\{\mu_X(x), \mu_Y(y)\}$
Prod - Prod*	$\mu_X(x) * \mu_Y(y)$	$\mu_X(x) + \mu_Y(y) - \mu_X(x) * \mu_Y(y)$
W - W*	$\text{Max}\{0, \mu_X(x) + \mu_Y(y) - 1\}$	$\text{Min}\{1, \mu_X(x) + \mu_Y(y)\}$

Tabla A.1: Definiciones de operadores borrosos.

y la de Mamdani [Mamdani, 1984], aunque existen otras como la Gödel, Goguen o la Willmot que se usan en determinados entornos por las propiedades asociadas. Su definición se realiza, al igual que en los operadores anteriores, utilizando las funciones de pertenencia. Así, la función de pertenencia $\mu_{X \longrightarrow Y}()$ de la implicación se puede calucular según se define en la Tabla A.2.

Luckasiewicz	$\text{Min}\{1, 1\text{- } \mu_X(x) + \mu_Y(y)\}$
Mamdani	$\text{Min}\{\mu_X(x), \mu_Y(y)\}$

Tabla A.2: Métodos habituales de implicación borrosa.

Con la definición de los operadores básicos se puede plantear un patrón de inferencia borrosa, o lo que es lo mismo, una versión difusa del tradicional *Modus Ponens* [Zadeh, 1988]. Es decir, el mecanismo para calcular el valor generado en y por una expresión del tipo "*Si x es X Entonces y es Y*", sabiendo que "x es X^*". Es decir, que no es exactamente X, que x es un valor difuso.

En la lógica bivaluada tradicional, el *Modus Ponens* se calcula mediante la expresión $XAND(X \longrightarrow Y)$, donde X equivale a "x es X" y $(X \longrightarrow Y)$, se calcula según una de las dos fórmulas anteriores. De esta forma, con la lógica tradicional, si X era Verdad y $(X \longrightarrow Y)$ era Verdad, se obtiene que Y era verdad. Aplicando ahora las fórmulas anteriores para el caso de que "x es X^*" (ni falso ni verdadero) se obtiene:

$$X^*AND(X \longrightarrow Y) \text{ Es igual a: } \text{Min}\{\mu_{X^*}(x), \text{Min}(\mu_X(x), \mu_Y(y))\}$$

Lo que es lo mismo, se obtendrá un valor de Y, ni falso ni verdadero, Y^*, definido por la función de pertenencia $\mu_{Y^*}(y)$ que se calculará como:

$$\mu_{Y^*}(y) = Supremo_x \{ \text{Min}\{\mu_{X^*}(x), \text{Min}(\mu_X(x), \mu_Y(y))\}$$

que desarrollado es:

$$\mu_{Y^*}(y) = Supremo_x \{ \text{Min } (\mu_Y(y), \text{Min}(\mu_{X^*}(x), \mu_X(x)))\}$$

donde $\text{Min}(\mu_{X^*}(x), \mu_X(x))$ se puede entender como el grado de "compatibilidad" entre "x es X^*" y "x es X".

Este sería el proceso de inferencia borrosa reflejado en la Figura A.1, que obtendría los conjuntos borrosos definidos sobre el dominio de discurso de las variables de salida que se muestra en la parte superior derecha en forma de trapecios de distinta altura. En principio, en un sistema borroso se ejecutan todas las reglas simultáneamente y todas ellas aportan un grado de activación, o de verdad a sus conclusiones. De este modo, la salida será igualmente borrosa, pues estará compuesta por conjuntos borrosos que representan las variables lingüísticas de salida, cada uno de ellos con un grado de activación distinto según el grado de "compatibilidad" de las premisas.

La forma de combinar ahora esas salidas para obtener una salida no borrosa dependerá del sistema que se haya elegido para dar soporte al razonamiento borroso. Esta es la última etapa que refleja la Figura A.1 y que se suele denominar "Proceso de Desborrosificación" (del inglés *Defuzzyfication*). Es decir, el proceso por el cual se obtienen las salidas numéricas deseadas.

El software utilizado en ABC^2 emplea un mecanismo denominado de "Centro de Gravedad". Este método trata cada regla por separado. Así, parte del valor de activación que cada regla induce en las etiquetas de la salidas λ_{Y_i} y calcula el final de la siguiente forma: sea C_{Y_i} el valor numérico representativo de cada etiqueta Y_i de la salida, (calculado como el centro de gravedad del trapecio que la define). Entonces la salida se calcula como la suma ponderada:

$$sum = (\sum \lambda_{Y_i} * C_{Y_i})/(\sum \lambda_{Y_i}).$$

En resumen, el sistema borroso toma una serie de valores numéricos como entradas, trabaja con ellas de forma lingüística, y obtiene valores numéricos para las salidas. Con las definiciones dadas, un diseñador solamente deberá aportar la definición de las etiquetas para las variables de entrada (los trapecios de la izquierda en la figura) y de salida (los de la derecha); y escribir las reglas utilizando las etiquetas previamente definidas.

Diseño de Reglas Borrosas

El diseño de un conjunto de reglas se puede considerar como una tarea delicada, donde influye la práctica y el dominio del entorno para el que se escribe el controlador. A

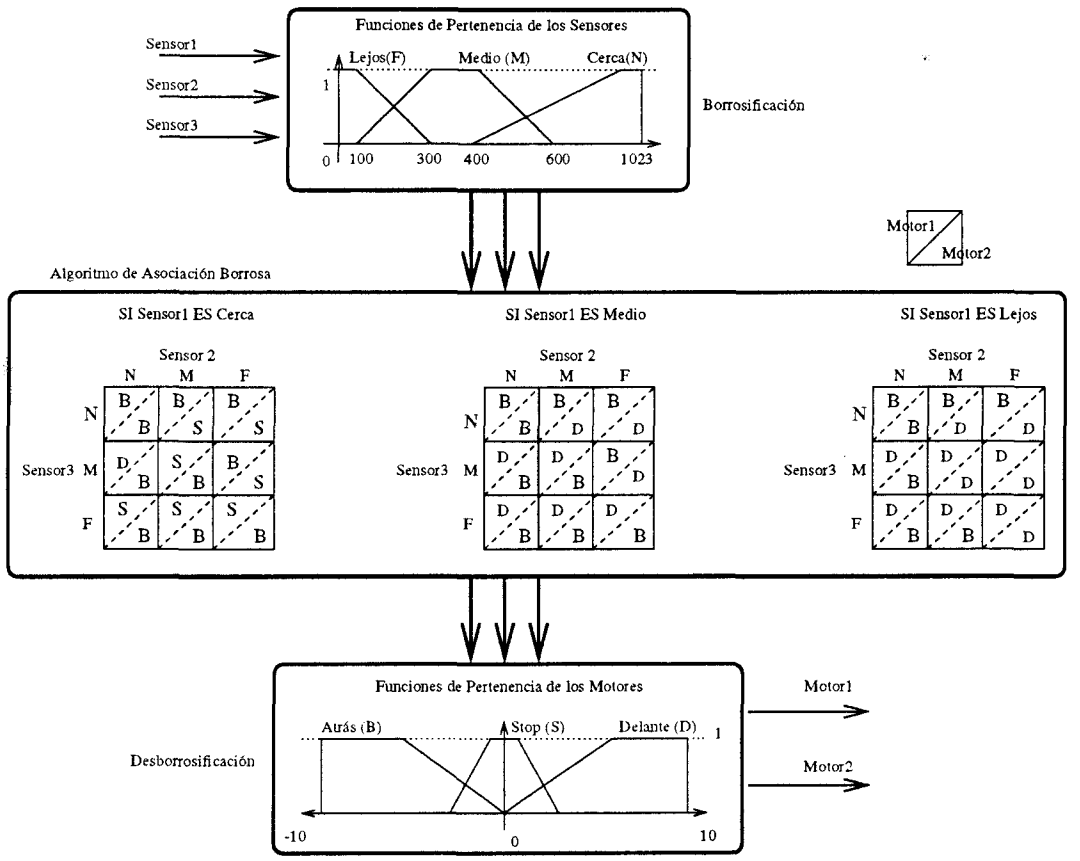

Figura A.2: Definición borrosa de la habilidad *Evitar_Obstáculos*.

modo de ejemplo, la Figura A.2 presenta un ejemplo de controlador sencillo para un robot que tenga tres sensores de proximidad y dos motores.

Para cada una de las variables de entrada, los sensores, se han definido tres conjuntos borrosos: Cerca, Medio y Lejos. La definición de dichas etiquetas se hace teniendo en cuenta las características físicas del robot. Así, el universo de discurso de dichas variables será el posible rango de valores devueltos por el robot, en este caso 1024 que van de 0 a 1023. El significado de las etiquetas tendrá que ver con su significado físico. Así, la variable cerca, por ejemplo se define como un trapecio comprendido entre los valores 400 y 1023, pero sólo se asigna el valor máximo de pertenencia a ese conjunto borroso a los valores comprendidos entre 950 y 1023. El diseño se ha realizado de esa forma teniendo en cuenta un estudio previo de la sensibilidad de los sensores.

Las etiquetas de los motores se muestran también en la figura. De nuevo la definición de las etiquetas responde a criterios lógicos. Por ejemplo, son simétricas, lo cual es lógico si los motores se pueden considerar iguales. El diseño de las reglas se muestra en forma de tres tablas de 9 casillas cada una. Dentro de cada casilla están las etiquetas

correspondientes a las variables de los motores. Por tanto las reglas se leen comenzando por la expresión condicional que tienen sobre ellas. Por ejemplo, la casilla superior izquierda de la primera tabla correspondería a la regla:

IF *sensor*1 **Es** *Cerca* **AND** *sensor*2 **Es** *Cerca* **AND** *sensor*3 **Es** *Cerca* **Entonces** *Motor*1 **Es** *Atrás* **AND** *Motor*2 **Es** *Atrás*

Se pueden aplicar simplificaciones que permiten acortar el formato de las reglas. Por ejemplo, las tres casillas superiores de la tabla pueden acortarse a:

IF *sensor*1 **Es** *Cerca* **AND** *sensor*2 **Es** *Cerca* **Entonces** *Motor*1 **Es** *Atrás*

No existen métodos formales que permitan averiguar si un conjunto de reglas produce el resultado deseado, tan solo la experimentación y el diseño cuidadoso pueden ayudar en ese sentido.

Por último, indicar que el software utilizado en ABC^2 se implementó en forma de biblioteca independiente, realizada utilizando el lenguaje C. Dicha librería permite definir reglas borrosas de forma simbólica utilizando un pequeño lenguaje de especificación de etiquetas y reglas [Blazquez *et al.*, 1997] que es el que se ha empleado en los ejemplos mostrados en esta memoria.

Apéndice B

Despejando entre dos

En el servidor Web de la RoboCup se encuentran disponibles las grabaciones de los tres partidos jugados por el equipo UC3M, que utilizó el modelo expuesto, durante la RoboCup97. La complejidad y el elevado número de los comportamientos desarrollados para la competición hacen imposible incluir un ejemplo completo del juego en esta memoria. Sin embargo, el dominio de la RoboCup es un dominio sobre el que se ha desarrollado una parte muy significativa del trabajo experimental de esta tesis. Por ello se ha considerado importante incluir en la presente memoria un ejemplo completo, aunque sea simple, de uso de ABC^2 en el dominio de la RoboCup.

La descripción de dicho ejemplo se refleja en este apéndice. Así, se muestra la configuración de los tres tipos de jugadores definidos en el ejemplo. En primer lugar se incluye la definición del portero, se incluye la definición de habilidades del delantero -encargado de llevar la bola desde el centro del campo hasta la portería contraria y chutar a puerta- y las de los dos defensas.

También se describe el formato de las reglas heurísticas empleadas, detallando el formato de definición empleado. Por último, se presentan los resultados del experimento.

Definición del experimento

La idea de este experimento es mostrar, de forma sencilla, el funcionamiento completo de ABC^2. Para ello se han definido dos equipos. El primero defiende y consta de tres jugadores, dos defensas que van a permanecer quietos en su posición simplemente mirando la bola y un portero. Este permanecerá en su posición hasta que le parezca que la pelota se encuentra lo suficientemente cerca. Entonces intentará salir a despejarla (en la versión oficial del simulador para la RoboCup'97 no existía la opción de coger la

pelota) hacia otro compañero. El equipo atacante está compuesto por un solo jugador con habilidades de llevar la pelota hacia la portería contraria y chutar a puerta en cuanto llegue cerca del área.

Cada uno de los jugadores en el dominio de la RoboCup debe implementarse como un proceso independiente que se comunica únicamente con el simulador vía sockets UDP. La versión de ABC^2 que se implementó para este dominio se realizó suponiendo que el binario fuese el mismo para todos los jugadores y que su especialización se realizaría mediante ficheros que enumerasen sus características: habilidades, relación entre ellas, prioridades heurísticas, etc. Esos ficheros son los que se muestran en este anexo.

En primer lugar se especifica la posición inicial (coordenadax, coordenaday y ángulo), la tolerancia, es decir, la distancia a la que va a considerar que se encuentra en una posición[1] y el nombre del jugador.

A continuación se especifican las habilidades que lo definen, dando su nombre y el peso inicial que se les da. Para mantener la similitud con el dominio real de la RoboCup se ha incluido el comportamiento `Ir_Posición` en todos los jugadores. Esto era indispensable en la competición real, pues se necesitaba para llevar a cada jugador a la posición inicial (en su propio campo) después de un gol. Otra restricción, esta vez de implementación, establece que las habilidades que aparezcan en la lista de necesidades de otra estén declaradas antes de poderse emplear en dicha lista. Esta lista comienza con el carácter "{" y termina con "}" y está formada por los nombres de las habilidades.

La siguiente lista de habilidades, que se describe con el mismo formato, representa las habilidades conocidas de sus compañeros. Es decir, las páginas amarillas. El formato de descripción indica en primer lugar el nombre del robot y a continuación una lista de sus habilidades.

Por último, se especifica cuál es la tarea inicial, es decir, el *habilidad* que se insertará en la agenda en primer lugar. Los ficheros de configuración admiten líneas de comentario (las que comienzan por un $*$) para facilitar su lectura.

Portero

El comportamiento básico del portero, como en los porteros reales, es quedarse en la portería mirando la bola. Para ello necesita saber dónde está la bola y estar en su posición. Es decir, la habilidad mirar la bola se puede ejecutar (*Preparado*) si

[1] Es necesario pues la información visual induce errores en la estimación de la posición.

está en su posición y sabe dónde está la bola. La acción que realizará (*Ejecutar*), es simplemente girar para mirar a la bola de frente. Por eso su lista de necesidades, según muestra la Formalización B.1, incluye las habilidades `Ir_Posición` y `Buscar_Bola` que se encargan de hacer que el jugador vaya a su posición asignada y que busque la bola, respectivamente. Cual de las dos se ejecuta antes depende del peso inicial, donde se ha considerado más importante que vaya a su posición, y de lo que decidan las reglas heurísticas en cada momento.

Formalización B.1 Inicialización del portero

```
* Parámetros iniciales (x, y, ang, tol, name)
 -50 0 0 3 portero
* Comportamientos
 Ir_Posición 0.7
 Buscar_Bola 0.6
 Mirar_Bola_Posición  0.75 {
     Ir_Posición
     Buscar_Bola }
 Salir 0.8 {
     Mirar_Bola_Posición }
     Despejar 0.9 {
     Salir }
 Ganar_Partido 1 {
     Despejar }
* Habilidades de sus compañeros:
 defensa-izquierdo: {
     Mirar_Bola
     Ir_Posición
     Chutar_Gol }
 defensa-derecho: {
     Mirar_Bola
     Ir_Posición
     Chutar_Gol }
* Ahora el comportamiento inicial
 Ganar_Partido
* Fin del fichero
```

Si la bola llega a estar cerca, donde cerca se define de forma difusa como una distancia menor de 15 m., entonces sale a por ella. Es decir, el *Ejecutar* de `Salir` realizará los cálculos necesarios para ir al encuentro de la bola. Si la alcanza la despeja, lo que se implementa como un intento de pasar a un compañero o chutar hacia una posición vacía, en dirección a la portería contraria, siempre que no esté viendo a ningún compañero.

Para conseguir el comportamiento deseado bastaría con definir unas heurísticas muy simples, como por ejemplo las mostradas en la Formalización B.2. Estas heurísticas se definen como reglas fuzzy. Esto implica, según lo explicado en el Anexo A, que hay que definir en primer lugar las variables relevantes. Para este ejemplo se han considerado como entradas la situación del juego `Situación_Juego`, la distancia a la bola `Distancia_Bola` y la distancia a la posición `Distancia_Posición` en la que se supone que debería estar el portero, que se define como el centro de la portería en los parámetros iniciales de la Formalización B.1.

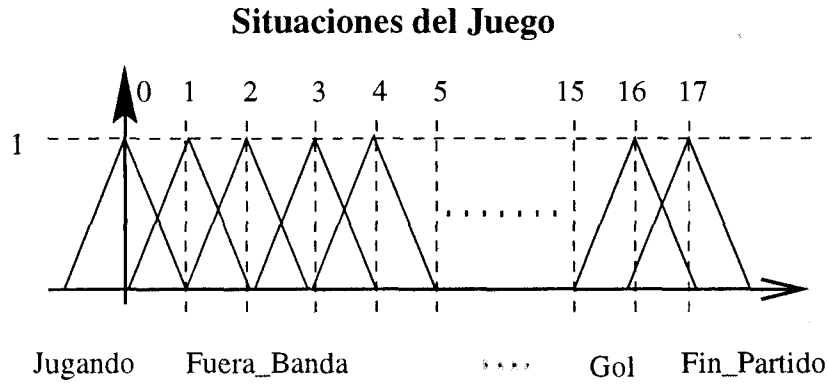

Figura B.1: Definición borrosa de las situaciones de juego.

Como salidas se definen todas las posibles habilidades de que disponga el jugador, en este caso `Despejar`, `Salir`, `Ganar_Partido`, `Mirar_Bola_Posición`, `Buscar_Bola` e `Ir_Posición`.

Para cada una de esas variables, siguiendo con el proceso de definición descrito en la Sección A, hay que definir las etiquetas lingüísticas. Por ejemplo, para las entradas referidas a la distancia (`Distancia_Bola` y `Distancia_Posición`) serán las etiquetas como *Muy_Cerca, Cerca, Lejos y Muy_Lejos*, para la situación del juego se definen una serie de situaciones como `Jugando`, `Descanso`, `Fuera_Banda`, ...hasta 18. Estas son situaciones booleanas, no borrosas (lo que se denomina en jerga borrosa, *crisp*). Para poder usarlas se definen de forma borrosa utilizando la definición Figura B.1.

De la misma forma, para las salida se definen etiquetas. En este caso son etiquetas que van a indicar cómo se modifica el peso de la habilidad concreta. Dicha definición se hace de forma borrosa y se corresponden con la Figura B.2.

No se incluye en este apéndice la definición formal de las variables ni la de las etiquetas por su farragosidad. Dicha definición implica detallar cada etiqueta por los cuatro valores del eje de abscisas que definen los trapecios de la Figura B.2. En el

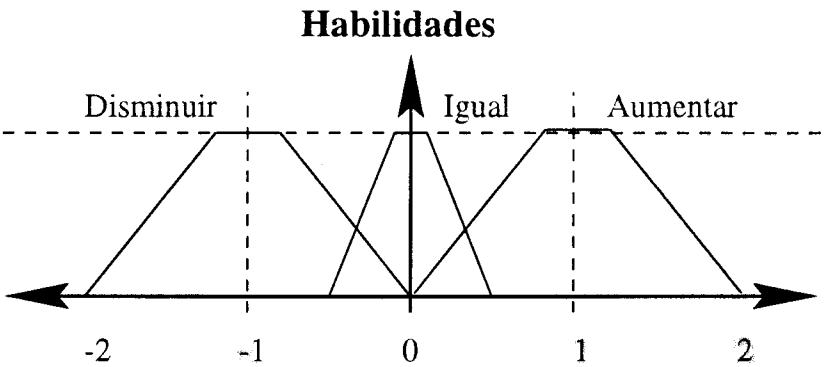

Figura B.2: Modificación de los pesos de las habilidades.

caso de los triángulos, los dos puntos centrales coinciden. Por ejemplo la definición de `Jugando` sería `-1 0 0 1`.

La definición de las etiquetas se realiza de forma heurística, esto es, el programador las debe fijar, al igual que las reglas, en función de las restricciones del entorno y a su experiencia. Así, las etiquetas referidas a la modificación de los pesos de las habilidades se hace de forma normalizada entre -1 y 1 de forma que sea coherente con los pesos iniciales asignados en las formalizaciones anteriores. Por ejemplo, la etiqueta `Disminuir` se define como `-2 -0.8 -0.4 0`.

Formalización B.2 Descripción de las heurísticas del portero

```
Situación_Juego es Pausa_Gol => Ir_Posición es Aumentar
Situación_Juego es Jugando => Ganar_Partido es Aumentar
Distancia_Posición es Lejos => Salir es Disminuir
Distancia_Posición es Muy_Lejos => Salir es Disminuir & Ir_Posición es Aumentar
Distancia_Bola es Cerca => Salir es Aumentar
Distancia_Bola es Muy_Cerca => Salir es Aumentar & Ir_Posición es Aumentar
```

Delantero

Si el comportamiento habitual de un portero es vigilar la bola desde su portería, un delantero nato debe tener el instinto simple de coger la pelota y chutar a puerta. La definición del jugador que se muestra en la Formalización B.3 realiza esta tarea.

Las habilidades no descritas anteriormente son `Buscar_Meta`, que se encarga de calcular la posición relativa de la portería contraria respecto de la posición actual y `Chutar_Gol`. Esta última sólo se podrá ejecutar cuando la bola esté suficientemente cerca y cuando además conozca donde está la portería contraria. El resultado de su

Formalización B.3 Descripción del "delantero"

```
* Parámetros iniciales (x, y, ang, tol, name)
 0 0 0 3 delantero
* Comportamientos
 Ir_Posición 1
 Buscar_Bola 0.6
 Buscar_Meta 0.5
 Ir_Bola 0.7 {
     Buscar_Bola }
 Chutar_Gol 0.8 {
     Ir_Bola
     Buscar_Meta }
 Ganar_Partido 0.9 {
     Chutar_Gol }
* Habilidades de sus compañeros:
 { }
* El comportamiento inicial
 Ganar_Partido
* Fin del fichero
```

ejecución será un chut, con la mayor fuerza posible, hacia la portería.

Defensa

Los defensas que se han definido para este simple experimento son un poco más particulares. Su misión es prácticamente no hacer nada. Simplemente deben quedarse en su posición mirando la bola y si ésta, casualmente, aparece a sus simulados pies, chutarla en dirección a la portería contraria.

La idea de tener unos defensas tan pasivos es debido a que el objetivo principal del experimento es comprobar que el portero es capaz de pasar la bola adecuadamente. Para definir el comportamiento de los defensas se hace uso de los comportamientos previamente definidos. Así, se define como su objetivo más importante chutar la bola, que necesita tenerla cerca para ejecutarse. Sin embargo, no se incluye la habilidad `Ir_Bola` entre sus necesidades, con lo que no será capaz de ir a buscarla. Solo podrá chutar cuando ocasionalmente, de forma oportunista, la bola aparezca cerca.

Ejemplo de funcionamiento

La figura B.3 muestra el comportamiento que se produce con las habilidades definidas anteriormente. La figura muestra la evolución mediante 6 imágenes que se deben

interpretar por filas, de izquierda a derecha.

Así, la primera imagen (esquina superior izquierda) muestra la situación inicial, con los cuatro jugadores en sus posiciones, tal y como se define en su formalización. La posición corresponde justo al instante en el que el delantero ha chutado a puerta.

Formalización B.4 Definición de los "defensas"

```
* Parámetros iniciales (x, y, ang, tol, name)
 -30 -10 0 4 defensa-izquierdo
* Comportamientos
 Ir_Posición 0.6
 Buscar_Bola 0.5
 Buscar_Meta 0.4
 Mirar_Bola 0.7 {
 Ir_Posición
     Buscar_Bola }
 Chutar_Gol 0.9 {
     Mirar_Bola
     Buscar_Meta }
 Ganar_Partido 1 {
     Chutar_Gol }
* Habilidades de sus compañeros:
 portero: {
     Mirar_Bola
     Ir_Posición
     Salir
     Despejar }
 defensa-derecho: {
     Mirar_Bola
     Ir_Posición
     Chutar_Gol }
* Ahora el comportamiento inicial
 Ganar_Partido
* Fin del fichero
```

La segunda imagen muestra cómo el delantero va avanzando hacia la portería contraria. La tercera, en la segunda fila, es el instante del chut a puerta del delantero. En la siguiente, el portero ya ha abandonado su posición y se dirige hacia la bola. En la quinta el portero ha atrapado la bola y decide pasarla al jugador de su izquierda. En la siguiente, la bola está a punto de alcanzar al defensa, que no hace nada por ir hacia ella. En la penúltima, el defensa ha recibido la bola y la chuta en la dirección contraria. La imagen final muestra cómo el defensa ya ha despejado la bola pero tampoco va tras ella.

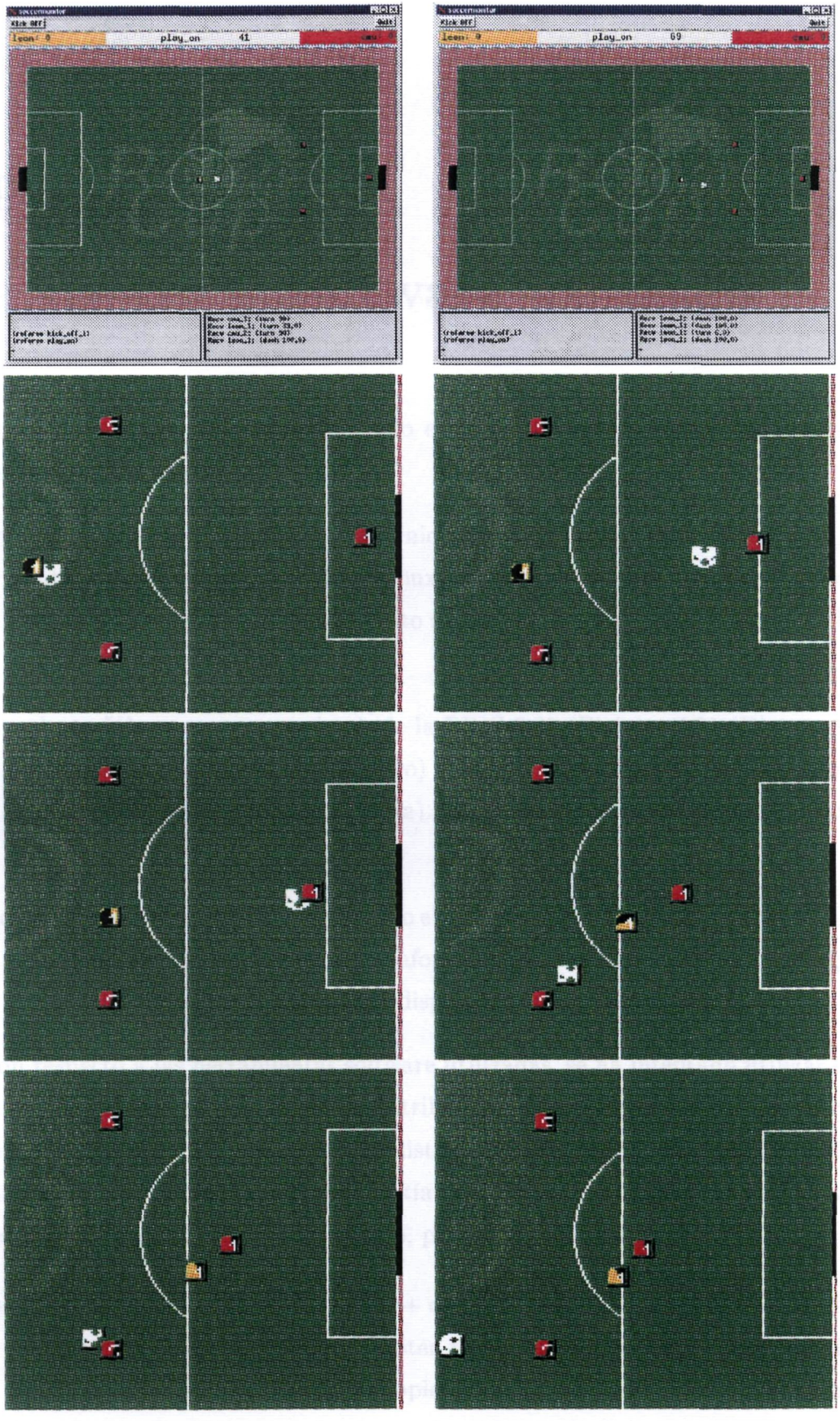

Figura B.3: Un ejemplo en el dominio de la RoboCup.

Apéndice C

Hardware y Software utilizado

Los sistemas físicos que se han empleado en el transcurso de este trabajo han sido, ordenados por tiempo de uso:

Ordenadores Personales de tipo 486 inicialmente y posteriormente Pentium, utilizando el sistema operativo GNU/Linux, una versión de Unix de libre distribución. Esta es la plataforma en la que tanto se ha escrito esta memoria como se ha desarrollado todo el software.

Mini-robots Khepera con versiones de la BIOS 3.01 (Pixie) y 4.01 (Dixie). Ambos son mini-robots (5.5 cm de diámetro) basados en el micro-controlador Motorola 68331 fabricados por K-Team (Suiza). Disponen de dos motores y de 8 sensores de infrarrojos.

Estaciones de trabajo SUN utilizando el sistema operativo Solaris/SPARC. Dichas estaciones se utilizaron como plataforma oficial de la RoboCup en Nagoya y también era la estación de trabajo disponible en Carnegie Mellon.

Con respecto a las herramientas software utilizadas, se ha intentado utilizar dentro de lo posible, herramientas de libre distribución. Es decir, aquéllas que permiten que su código fuente sea conocido y redistribuido con muy pocas restricciones. La razón principal de ello es la mayor garantía que dichas aplicaciones ofrecen. Las más relevantes de entre las utilizadas han sido, por orden alfabético:

gcc y g++ son compiladores de C y C++ de GNU. Son unos de los compiladores más extendidos en el mundo UNIX y existen tanto en plataformas libres, GNU/Linux, NetBSD, etc.; como en sistemas propietarios como Solaris. Por ello la migración del software de uno a otro sistema no ha sido en absoluto costosa.

gdb es el depurador de GNU, que unido al editor XEmacs permite disponer de un entorno gráfico de depuración. En el transcurso de los experimentos se han utilizado otros depuradores, como **ddd**, pero sin duda, **gdb** ha sido el más empleado.

XEmacs es un editor extensible y programable con una interfaz de usuario gráfica y que está especialmente bien adaptado a la programación. Existen multitud de extensiones (modos) para programar en C, C++, escribir en LaTeX (como se ha escrito esta memoria), visualizar páginas HTML, etc.

LaTeX es un sistema para la creación de documentos basado en TeX que permite la escritura sencilla de textos. Ha sido la herramienta empleada para la escritura de esta memoria.

GNU/Linux , sistema operativo tipo Unix originalmente desarrollado por Linus Torvald y que se ha consagrado como uno de los sistemas operativos, después de Windows 95[1], más utilizados del mundo. El hecho de añadir el calificativo GNU al nombre típico de Linux, quiere indicar que bajo el nombre GNU/Linux se están englobando múltiples herramientas no originales de Linux, la mayoría desarrolladas bajo el amparo de GNU, que hacen realmente utilizable este sistema operativo.

make es una herramienta para automatizar el proceso de compilación. Se ha empleado en concreto la versión de make de GNU.

SimDAI es el simulador de robots móviles (tipo Khepera) desarrollado en el Laboratorio de Agentes Inteligentes de la Universidad Carlos III de Madrid.

SoccerServer es el simulador oficial de la RoboCup desarrollado por Itsuki Noda en el ETL. Los experimentos descritos se realizaron con la versión oficial de la RoboCup'97.

xfig es una herramienta de dibujo, orientada a objetos, que se ha utilizado para la realización de la mayoría de los gráficos y diagramas de este texto.

XFree86 implementación del sistema X-Window que permite disponer de una interfaz gráfica distribuida sobre GNU/Linux.

[1]Marca Registrada de Microsoft.

Todo este software es libre, lo que implica que su código fuente está disponible, y aunque está protegido bajo diferentes licencias (normalmente GPL), en su mayoría puede distribuirse sin dificultad. De hecho, todas estas herramientas pueden encontrarse sin ningún problema en la red.

Apéndice D

Referencias en la red

Durante siglos, la forma de extender y contrastar el conocimiento ha sido la publicación de libros, consagrando la referencia y la comparación con otros autores como una parte imprescindible del proceso de generación científica.

La presentación de los trabajos de investigación en las revistas científicas y la proliferación de conferencias y talleres especializados permitió acelerar el proceso de difusión, discusión y aceptación de los resultados científicos por lo que se añadieron al capítulo de referencias bibliográficas.

Por último, la aparición de las modernas redes de comunicación permite hoy en día una mayor rapidez y comodidad en el acceso a la información científica y por tanto, cualquier obra debe incluir una referencia a dichas fuentes.

Una lista no exhaustiva de referencias relativas a esta tesis doctoral que se puede encontrar en la red, es la que se muestra a continuación. Se ha organizado en cuatro categorías y se ha utilizado la sintaxis de las URLs para su definición:

Modelos

BB1 :

http://ksl-web.standford.edu

CIRCA :

http://ai.eecs.umich.edu/people/marbles/circa/CIRCA.html

PRODIGY :

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/prodigy/Web/prodigy-home.html

PRS :

http://www.ai.sr.com/~prs

RAP :

http://www.cs.uchicago.edu/

SOAR :

http://www.isi.edu/soar

STEAM :

http://www.isi.edu/soar/tambe/steam/steam.html

TCA :

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/TCA/release/tca.home

Robots

Cog :

http://www.ai.mit.edu/projects/cog

Herbert :

http://www.ai.mit.edu/projects/mobile-robots/veterans.html

Khepera :

http://lamiwww.epfl.ch/Khepera

Rhino :

http://www-cs.uni-bonn.de/~rhino

Spinoza :

http://www.cs.ubc.ca/nest/lci/robots/spinoza.html

Xavier :

http://www.cs.cmu.edu/~Xavier.html

Simuladores

BeeSoft :

http://www.rwi.com/software/index.html

JavaSoccer :

http://www.gatech.edu/grads/Tucker.Balch/JavaBots

MICROB (Robotic soccer) :

http://www-poleia.lip6.fr/~comiriad/microb/microb_soft.html

SimDAI :

http://www.gsyc.inf.uc3m.es/~vmo/SimDAI/

SoccerServer :

http://ci.etl.go.jp/noda/~soccer/server.html

MAGSY :

ftp://ftp.dfki.uni-sb.de/pub/MAGSY/MAS-MARS/system

NHDE :

http://www.robots.com/soft.html

Webots :

http://diwww.epfl.ch/lami/team/michael/webots

RoboCup

RoboCup :

http://www.RoboCup.org

AT-Humboldt-97 : (Campeones de la RoboCup'97, Simulator League)

http://www.informatik.hu-berlin.de/Institut/struktur/ki/RoboCup97/index_e.html

Humboldt University of Berlin, Alemania (Hans-Dieter Burkhard, Jan Wendler, et al.).

CMUnited : (Campeones de la RoboCup'97, Small League)

http://www.cs.cmu.edu/~mmv/

Carnegie Mellon University, USA (Manuela Veloso, Peter Stone, Kwun Han, Sorin Achim, and Michael Bowling).

Dreamteam : (Campeones en la RoboCup'97, Middle-Size Robot League)

http://www.isi.edu/isd/futbot5/

University of Southern California, Information Science Institute, USA (Wei-Min Shen, et al.).

The Official RoboCup Simulator :

http://ci.etl.go.jp/~noda/soccer/server.html

Aplicación a la enseñanza :

http://www.ida.liu.se/~jacma/official/aip.html

Curso de Programación para Inteligencia Artificial (utilizando RoboCup), impartido por Silvia Coradeschi y Jacek Malec, Universidad de Linkoeping (Suecia).

Apéndice E

Glosario de Términos y Acrónimos

Este apéndice incluye breves descripciones de algunos términos con diversos significados en castellano, así como de otros traducidos del inglés y acrónimos utilizados en esta memoria.

Términos

Acto (ABC^2) Acción potencial que el agente está considerando realizar.

Agenda (ABC^2) Estructura de datos que gestiona el conjunto de actos que un agente está considerando en un momento determinado.

Agente (ABC^2) Entidad computacional capaz de tomar sus decisiones sin la intervención de ningún operador humano.

Habilidad (ABC^2) Controlador capaz de realizar una determinada acción.

Heurísticas (ABC^2) Mecanismo para asignar el grado de prioridad a los actos contenidos en la agenda según la situación existente.

Páginas Amarillas (ABC^2) Información sobre las habilidades de otros agentes.

Planificación clásica Conjunto de técnicas de generación de planes basadas en la búsqueda sobre espacios de estados definidos generalmente de forma simbólica.

Planificación Oportunista Grupo de sistemas de planificación que especifican para cada operador las circunstancias en las que puede aplicarse.

RoboCup Término que engloba las competiciones y conferencias relativas al problema del fútbol entre robots.

Sistemas Reactivos Conjunto heterogéneo de métodos para la toma de decisiones, caracterizado por utilizar únicamente la información local disponible.

Acrónimos

AAAI: *American Asociation for Artificial Intelligence*

Es probablemente la organización dedicada a la inteligencia artificial más importante del mundo.

CIRCA: *Cooperative Intelligent Real-Time Control Architecture*

Arquitectura para el control de robots diseñada por el grupo de Edmund H. Durfee.

ETL: *ElectroTechnical Laboratory*

Centro perteneciente al gobierno japonés dedicado a investigar, entre otros campos, en sistemas multi-agentes para lo que utilizan el dominio del fútbol entre robots. Son los encargados de desarrollar el simulador oficial de la RoboCup.

GNU: *Gnu is Not Unix*

Definición recursiva que agrupa un conjunto muy importante de software libre, incluyendo por ejemplo aplicaciones como gcc, gdb, emacs, etc.

GPL: *GNU Public License*

Licencia, de uso público, que protege al software de libre distribución realizado por GNU. Esta licencia garantiza que el código fuente de las aplicaciones permanece libre.

GPS: *General Problem Solver*

Primer sistema de resolución simbólica de problemas. Se basa en el análisis de medios-fines y fue propuesto por Allen Newell y Herbert Simon en los años 60.

GSyC: *Grupo de Sistemas y Comunicaciones*

Grupo encargado de la docencia de sistemas operativos, transmisión de datos, sistemas distribuidos, sistemas tolerantes a fallos y programación de la Universidad Carlos III de Madrid. Es el grupo en el que se encuentra integrado actualmente el autor de este trabajo.

IEEE: *Institute of Electrical and Electronical Engineers*

Asociación norteamericana de ingenieros que fomenta el desarrollo de la ciencia y la tecnología mediante la publicación de numerosas revistas, libros, organización de conferencias, cursos, etc.

IA: *Inteligencia Artificial*

Es el acrónimo que se ha empleado en esta tesis para designar la traducción al castellano del término *Artificial Intelligence* que designa el área en la cual se enmarca este trabajo.

InteRRaP: *Integration of Reactive behavior and Rational Planning*

Arquitectura para el control de múltiples robots propuesta por Jörg Müller et al.

LAI: *Laboratorio de Agentes Inteligentes*

Grupo de investigación del Departamento de Informática de la Universidad Carlos III de Madrid al que perteneció el autor durante la mayor parte del desarrollo de esta tesis.

RoboCup: *Robot World Cup Soccer Games and Conferences*

Constituye un intento de promocionar los campos de la robótica y la inteligencia artificial proporcionando un problema estándar, el fútbol, donde se pueden integrar gran cantidad de tecnologías.

SimDAI: *Simulator for Distributed Artificial Intelligence*

Simulador de robots autónomos desarrollado en el LAI.

TAP: *Test-Action Pair*

Sistema para la definición de controladores en la arquitectura CIRCA.

TCA: *Task Control Architecture*

Arquitectura de control de robots desarrollada por Reid Simmons et al. y que aborda el control integral del robot.

TCP/IP: *Transmission Control Protocol / Internet Protocol*

Indentifican en general una familia de protocolos que es la base de Internet. Los más conocidos son IP, nivel de red; TCP, para el transporte fiable y UDP para transporte no fiable.

TR: (*Teleo-Reactive*)

Son los programas propuestos por Nils Nilsson como sistema de control de agentes autónomos.

UC3M: *Universidad Carlos III de Madrid*

Acrónimo que suele identificar a la Universidad donde se ha desarrollado fundamentalmente el presente trabajo.

UDP: *User Datagram Protocol*

Protocolo de transporte no fiable, orientado a datagramas, que forma parte de la familia de protocolos TCP/IP y que es el empleado en el simulador de la RoboCup.

UM-PRS: *University of Michigan Precedural Reasoning System*

Sistema de Razonamiento Procedural (PRS) de la Universidad de Michigan (UM), que se puede definir como una arquitectura de tiempo-real para el control de múltiples agentes.

URL: *Uniform Resource Locator*

Representación normalizada para las direcciones de internet que incluye información sobre la máquina, el puerto y el protocolo que se usa para acceder a un servicio.

WWW: *World Wide Web*

Servicio que proporciona una gran colección de servidores en Internet, que ofrecen información multimedia interactiva y enlazada de forma hipertextual.

Bibliografía

[Agre and Chapman, 1990] Philip E. Agre y David Chapman. What are plans for? En Pattie Maes, editor, *New architectures for autonomous agents: task-level decomposition and emergent functionality.* MIT Press, Cambridge, Massachusetts (USA), 1990.

[Arkin, 1992] Ronald C. Arkin. Cooperation without communication: Multiagent schema-based robot navigation. *Journal of Robotic Systems*, 9(3):351–364, 1992.

[Arkin, 1993] Ronald C. Arkin. Survivable robotic systems: Reactive and homeostatic control. En Mo Jamshidi y Patrick J. Eicker, editores, *Robotics and Remote Systems for Hazardous Environments.* Prentice Hall, 1993.

[Asama *et al.*, 1991] Hajime Asama, Maki K. Habib, Isao Endo, Koichi Ozaki, Akihiro Matsumoto, y Yoshiki Ishida. Functional distribution among multiple mobile robots in an autonomous and descentralized robot system. En *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, páginas 1921–1926, Sacramento-California (USA), 1991.

[Asama *et al.*, 1992] Hajime Asama, Koichi Ozaki, Akihiro Matsumoto, Yoshiki Ishida, y Isao Endo. Development of task assignment system using communication for multiple autonomous robots. *Journal of Robotics and Mechatronics*, 4(2):122–127, 1992.

[Asimov, 1950] Isaac Asimov. *I robot.* Gnome Press. Doubleday desde 1961, 1950.

[Barahona, 1998] Jesús G. Barahona. *Arquitectura de comunicaciones distribuida para procesos replicados.* Tesis Doctoral, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid, Febrero 1998.

[Blazquez *et al.*, 1997] José Manuel Blazquez, José Manuel Molina, y Vicente Matellán. Control de agentes mediante lógica borrosa. Tutorial rama española del IEEE, Madrid, España, Abril 1997.

[Blum and Furst, 1995] Avrim L. Blum y Merrick L. Furst. Fast planning through planning graph analysis. En Chris S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volumen 2, páginas 1636–1642, Montreal (Canadá), Agosto 1995. Morgan Kaufmann.

[Blythe, 1994] Jim Blythe. Planning with external events. En *Conference on Uncertainty in AI*, 1994.

[Blythe, 1996] Jim Blythe. Event-based decomposition for reasoning about external changes in planner. En *Conference on Uncertainty in AI*, páginas 27–34, Edinbourgh, 1996.

[Bond and Gasser, 1988] Alan H. Bond y Les Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, 1988.

[Bowling *et al.*, 1996] Mike Bowling, Peter Stone, y Manuela Veloso. Predicitive memory for an inaccessible environment. En *Working Notes of the IROS-96 Workshop on RoboCup*, 1996.

[Braitenberg, 1984] Valentino Braitenberg. *Vehicles. Experiments in Synthetic Psychology.* MIT Press, Cambridge, MA (USA), 1984.

[Brooks, 1986] Rodney A. Brooks. A roboust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, Marzo 1986.

[Brooks, 1991a] Rodney A. Brooks. Artificial life and real robots. En *Proceedings of the First European Conference on Artificial Life*, páginas 3–10, Paris, France, 1991.

[Brooks, 1991b] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, (47):139–159, 1991.

[Caloud *et al.*, 1990] Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape, y Mark Yim. Indoor automation with many mobile robots. En *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, páginas 67–72, Tsuchiura, Japón, 1990.

[Capek, 1921] Karel Capek. *R.U.R. Rossom's Universal Robots.* 1921.

[Ch'ng and Padgham, 1997] Simon Ch'ng y Lin Padgham. Team description: Royal Melbourne Knights. En *Workshop on RoboCup. Fifteenth International Joint Conference on Artificial Intelligence IJCAI-97*, páginas 125–128, Nagoya, Japan, Agosto 1997.

[Cohen and Perrault, 1986] Philip R. Cohen y C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, RA-2(3):177–212, 1986.

[Cohen *et al.*, 1990] Paul Cohen, Michael Greenberg, David Hart, y Adele Howe. Real-time problem solving in the phoenix enviroment. Informe Técnico COINS Technical Report 90-28, University of Massachusetts at Amherst, 1990.

[Collins and Balch, 1997] Tom Collins y Tucker Balch. Teaming up: Georgia Tech's multirob. En *Proceedings of AAAI-97*, 1997.

[Connell, 1990] Jonathan H. Connell. *Minimalist Mobile Robotics: A Colony-style Architecture for a Mobile Robot.* Academic Press, Cambridge, MA, 1990.

[Cook, 1993] Diane J. Cook. Using analytic and genetic methods to learn plans for mobile robots. En *Proceedings of the SPIE Conference on the Applications of Artificial Intelligence*, páginas 327–336, 1993.

[Cooper, 1995] Mark G. Cooper. Evolving a rule-based fuzzy controller. *Simulation*, 65(1), 1995.

[Coradeschi and Lars, 1997] Silvia Coradeschi y Karlsson Lars. A role-based decision-mechanism for teams of reactive and coordinating agents. En *Workshop on RoboCup. Fifteenth International Joint Conference on Artificial Intelligence IJCAI-97*, páginas 37–44, Nagoya, Japan, Agosto 1997.

[Coradeschi *et al.*, 1996] Silvia Coradeschi, Karlsson Lars, y Anders Törne. Intelligent agents for aircraft combat simulation. En *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida, 1996.

[Currie and Tate, 1991] Ken Currie y Austin Tate. O-plan: the Open Planning Architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[Deneubourg *et al.*, 1990] Jean-Louis Deneubourg, Simon Goss, Giulio Sandini, Federico Ferrari, y Paolo Dario. Self-organizing collection and transport of objects in unpredictable environments. En *Japón-USA Symposium on Flexible Automation*, páginas 1093–1098, 1990.

[Drogoul and Ferber, 1992] Alexis Drogoul y Jacques Ferber. From thumb to the dockers: Some experiments with foraging robots. En *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, páginas 451–459, 1992.

[Drummond *et al.*, 1994] Mark Drummond, John Bresina, y Keith Swanson. Just-in-case scheduling. En *Proceedings of AAAI-94*, Seattle, Washington, 1994.

[Durfee *et al.*, 1997] Edmund H. Durfee, Kang G. Shin, y Ella M. Atkins. Detecting and reacting to unplanned-for world states. En *Proceedings of the AAAI-97*, páginas 571–576, 1997.

[Fikes and Nilsson, 1971] Richard E. Fikes y Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Finin *et al.*, 1994] Tim Finin, Don McKay, Rich Fritzson, y Robin McEntire. *Knowledge Building and Knowledge Sharing*, capítulo KQML: An information and Knowledge Exchange Protocol. Ohmsha and IOS Press, 1994.

[Firby, 1994] James R. Firby. Task networks for controlling continuous processes. En *Proceedings of the Second International Conference on AI planning systems*, 1994.

[Firby, 1996] James R. Firby. Modularity issues in reactive planning. En *Proceedings of the Third International Conference on AI Planning Systems*, páginas 78–85, Edinburgh, Scotland, Mayo 1996.

[Fukuda and Ueyama, 1994] Toshio Fukuda y Tsuyoshi Ueyama. *Cellular Robotics and Micro Robotic System.* World Scientific, 1994.

[García, 1997] Francisco Serradilla García. *Arquitectura cognitiva basada en el gradiente sensorial y su aplicación a la robótica móvil.* Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Madrid, 1997.

[García-Martínez and Borrajo, 1997] Ramón García-Martínez y Daniel Borrajo. Planning, learning, and executing in autonomous systems. En Sam Steel, editor, *Recent Advances in AI Planning. 4th European Conference on Planning, ECP'97*, número LNAI 1348 in Lecture Notes in Artificial Intelligence, páginas 208–220, Toulouse, Francia, Septiembre 1997. Springer-Verlag.

[Georgeff and Lansky, 1986] Michael P. Georgeff y Amy L. Lansky. Procedural knowledge. En *Proceedings of IEEE*, volumen 10, páginas 1383–1398, Octubre 1986.

[Giráldez and Borrajo, 1997] José I. Giráldez y Daniel Borrajo. A distributed model for the combination of heterogeneous knowledge based agents. En J. Hunt y R. Miles, editores, *Proceedings of Expert Systems 97, The 17th SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, Research and Development in Expert Systems, páginas 175–184, Cambridge, England, December 1997. The British Computer Society. Specialist Group on Expert Systems (SGES), SGES Publications.

[Goldberg and Holland, 1988] David E. Goldberg y John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2/3):95–101, Octubre 1988.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in search, optimization and machine learning.* Addison-Wesley, U.S.A., 1989.

[González,] Antonio González. A learning methodology in uncertain and imprecise environments. *International Journal of Intelligent Systems*, 19:357–371.

[González *et al.*, 1994] Antonio González, Raul Pérez, y J.L. Verdegay. Learning the structure of a fuzzy rule: A genetic approach. *Fuzzy Systems and Artificial Intelligence*, 3(1):57–70, 1994.

[Hayes-Roth *et al.*, 1979] Barbara Hayes-Roth, Fredrick Hayes-Roth, Stan Rosenschein, y Stephanie Cammarata. Modeling planning as an incremental, opportunistic process. En *Proc. of the 6th IJCAI*, páginas 375–383, Tokio, Japan, 1979.

[Hayes-Roth, 1985] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[Hayes-Roth, 1992] Barbara Hayes-Roth. Opportunistic control of action in intelligent agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1575–1586, 1992.

[Hoffmann and Pfister, 1994] Frank Hoffmann y Gerd Pfister. Automatic design of hierarchical fuzzy controller using genetic algorithms. En *Proceedings of the EUFIT 94*, Aachen, Germany, 1994.

[Ishida *et al.*, 1994] Yoshiki Ishida, Hajime Asama, S. Tomita, Koichi Ozaki, Akihiro Matsumoto, y Isao Endo. Functional complement by cooperation of multiple autonomous robots. En *Proceedings of the 1994 International Conference on Robotics and Automation*, páginas 2476–2481, 1994.

[Itsuki, 1995] Noda Itsuki. Soccer server: A simulator for RoboCup. En *Proceedings of the AI-Symposium 95: Special Session on RoboCup*, páginas 29–34. Japanese Society for Artficial Intelligence, 1995.

[Kaelbling, 1987] Leslie P. Kaelbling. An architecture for intelligent reactive systems. En M. Georgeff y A. Lansky, editores, *Reasoning about plans and actions*, páginas 395–410. Morgan Kaufmann, San Mateo, California, 1987.

[Karr, 1991] Chuck Karr. Applying genetics to fuzzy logic. *AI Expert*, páginas 38–43, Marzo 1991.

[Koza, 1991] John R. Koza. Evolving emergent wall following robotic behavior using the genetic programming paradigm. En F.J. Varela y P. Bourgine, editores, *Toward a practice of autonomus systems. Proceedings of the First European Conference on Artificial Life*, páginas 110–119, Cambridge, MA, 1991. MIT Press and Bradford Books.

[Kube and Zhang, 1990] Ronald Kube y Hong Zhang. Collective robotic intelligence. En *Proceedings of the Second International Workshop on Simulation of Adaptive Behavior*, páginas 460–468, 1990.

[Lee and Takagi, 1993] Michael A. Lee y Hideyuki Takagi. Integrating design stages of fuzzy systems using genetic algorithms. En *Proceedings of the Second International Conference on Fuzzy Systems*, páginas 612–617, 1993.

[Lehman *et al.*, 1996] Jill Fain Lehman, John E. Laird, y Paul S. Rosenbloom. A gentle introduction to SOAR, an architecture for human cognition. En *Invitation to Cognitive Science*, volumen 4. MIT Press, 1996.

[Llavorí, 1996] Rafael Berlanga Llavorí. *STRPLAN: Planificación multi-agente en dominios estructurados.* Tesis Doctoral, Universidad de Valencia. Facultad de Física, Septiembre 1996.

[Lucas, 1977] George Lucas. Star wars. Film, 1977.

[Macworth, 1993] Alan K. Macworth. *Computer Vision: System, Theory, and Applications*, capítulo On Seeing Robots, páginas 1–13. World Scientific Press, Singapore, 1993.

[Maes and Brooks, 1990] Pat Maes y Rodney Brooks. Learning to coordinate behaviors. En *Proccedings of the Eighth National Conference on Artificial Intelligence*, páginas 796–802, San Mateo, CA, 1990. Morgan Kaufmann.

[Mamdani, 1984] Ebrahim H. Mamdani. An analysis of formal logics as inference mechanism on expert systems. *International Journal of Man-Machine Studies*, 21:1–13, 1984.

[Mataric, 1992] Maja Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, Junio 1992.

[Matellán and Borrajo, 1997] Vicente Matellán y Daniel Borrajo. An agenda based multi-agent architecture. En *Workshop on RoboCup. Fifteenth International Joint Conference on Artificial Intelligence IJCAI-97*, páginas 121–124, Nagoya, Japan, August 1997.

[Matellán *et al.*, 1995a] Vicente Matellán, José Manuel Molina, y Camino Fernández. Learning fuzzy behaviors for autonomous robots. En *Fourth European Workshop on Learning Robots*, páginas 45–50, Karlsruhe, Germany, December 1995.

[Matellán *et al.*, 1995b] Vicente Matellán, José Manuel Molina, y Camino Fernández. Fusion of fuzzy behaviors for autonomous robots. En *Proceedings of the Third International Symposium on Intelligent Robotic Systems*, páginas 157–164, Pisa, Italia, 1995.

[Matellán *et al.*, 1996] Vicente Matellán, José Manuel Molina, y Lorenzo Sommaruga. Fuzzy multi-agent interaction. En *Proccedings of the IEEE Conference on Systems, Man and Cibernetics*, páginas 1950–1955, Beijin, China, Oct 1996.

[Mayo, 1996] Natalia Mayo. Planificación multinivel. PFC, Universidad Carlos III de Madrid, Abril 1996.

[McAllester and Rosenblitt, 1991] David McAllester y David Rosenblitt. Systematic nonlinear planning. En *Proceedings of AAAI-91*, páginas 634–639, 1991.

[McFarland and Bösser, 1993] David McFarland y Thomas Bösser. *Intelligent Behavior in Animals and Robots.* Complex Adaptive Systems. MIT Press, 1993.

[Mckerrow, 1991] Phillip John Mckerrow. *Introduction to robotics.* Addison-Wesley Publishing Company, 1991.

[Miller *et al.*, 1960] George Armitage Miller, Eugene Galanter, y Karl Pribram. *Plans and the structure of behavior.* Holt, New York, 1960.

[Molina *et al.*, 1996] José Manuel Molina, Vicente Matellán, y Camino Fernández. Fuzzy decision making. En *Proceedings of Information Technologies ITHURS'96*, volumen 2, páginas 191–196, León, Spain, July 1996.

[Molina, 1997] José Manuel Molina. *Técnicas multiagente de cooperación para la gestión coordinada de multisensores.* Tesis Doctoral, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid, Julio 1997.

[Mondada *et al.*, 1993] Francesco Mondada, Edoardo Franzi, y Paolo Ienne. Mobile robot miniaturisation: A tool for investigation in control algorithms. En *Proceedings of the third International Symposium on Experimental Robotics*, Kyoto, Japan, 1993.

[Muller and Pischel, 1994] Jorg Muller y Markus Pischel. Modelling interacting agents in dynamic environments. En A. Cohn, editor, *ECAI 94, 11th European Conference on Artificial Intelligence.* John Wiley & Sons Ltd., 1994.

[Muslea, 1997] Ion Muslea. SINERGY: A linear planner based on genetic programming. En Sam Steel, editor, *Recent Advances in AI Planning. 4th European Conference on Planning, ECP'97*, número LNAI 1348 in Lecture Notes in Artificial Intelligence, páginas 312–324, Toulouse, France, Septiembre 1997. Springer-Verlag.

[Musliner *et al.*, 1993] David J. Musliner, Edmund H. Durfee, y Kang G. Shin. Circa: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1561–1574, 1993.

[Musliner *et al.*, 1994] David J. Musliner, James A. Hendler, Ashok K. Agrawala, Edmund H. Durfee, y Jay K. Strosnider. The challenges of real time artificial intelligence. Informe Técnico CS-TR-3290, Univesity of Maryland, 1994.

[Neisser, 1967] Ulric Neisser. *Cognitive Psicology.* Prentice Hall, New York, 1967.

[Newell and Simon, 1956] Allen Newell y Herbert Simon. The logic theory machine: A complex information processing system. *IRE Transactions on Information Theory*, páginas 61–79, 1956.

[Newell and Simon, 1963] Allen Newell y Herbert Simon. GPS: A program that simulates human thought. En E.A. Feigenbaum y J. Feldman, editores, *Computers and Thought*, páginas 279–293. McGraw-Hill, New York, 1963. Also in Readings in Planning.

[Newell and Simon, 1972] Allen Newell y Herbert Simon. *Human Problem Solving.* Prentice-Hall, 1972.

[Nilsson, 1969] Nils J. Nilsson. A mobile automaton: an application of artificial intelligence techniques. En *Proceedings of the IEEE International Conference on Robotics and Automation*, 1969.

[Nilsson, 1994] Nils J. Nilsson. Teleo-Reactive programs for agent control. *Artificial Intelligence Journal*, (1):139–158, Enero 1994.

[Noda, 1995] Itsuki Noda. Soccer server: A simulator of robocup. En *Proceedings of AI Symposium'95.* Japanese Society for Artificial Intelligence, dec 1995.

[Noreils, 1993] Fabrice R. Noreils. Towards a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98, 1993.

[Pavlov, 1927] Ivan P. Pavlov. *Conditioned Reflexes.* Humphrey Milford, 1927.

[Penberthy and Weld, 1992] J. S. Penberthy y D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. En *Proceedings of KR-92*, páginas 103–114, 1992.

[Pérez, 1991] M. Alicia Pérez. Mulit-agent planning in PRODIGY. Informe Técnico CMU-CS-91-139, School of Computer Science, Carnegie Mellon University, 1991.

[Pin and Watanabe, 1993] Francois G. Pin y Yutaka Watanabe. Driving a car using reflexive fuzzy behaviors. En *Proceedings of the Second International Conference on Fuzzy Systems*, San Francisco, USA, 1993.

[Premvuti and Yuta, 1990] Suparek Premvuti y Shin'ichi Yuta. Considerations on the cooperation of multiple autonomous mobile robots. En *Proceedings of 1990 IEEE International Workshop on Intelligent Robots and Systems IROS*90, páginas 59–63. IEEE / RSJ IROS, 1990.

[Quinlan, 1993] Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[Reignier, 1994] Patrick Reignier. Fuzzy logic techniques for mobile robot obstacle avoidance. *Robotics and Autonomous Systems*, 12:143–153, 1994.

[Ribagorda *et al.*, 1996] Arturo Ribagorda, Alfonso Calvo Orra, y Miguel Angel Gallardo. *Seguridad en Unix.* Paraninfo, 1996.

[Rumbaugh *et al.*, 1991] James Rumbaugh, Michael Blah, Willian Premerlani, Frederick Eddy, y Willian Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall International Editions, New Jersey, USA, 1991.

[Sacerdoti, 1973] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. En *Proceedings of the Third International Joint Conference on Artificial Intelligence*, páginas 412–430, Standford, California, 1973.

[Sacerdoti, 1977] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, New York, 1977.

[Saffiotti and Ruspini, 1993] Alessandro Saffiotti y Enrique H. Ruspini. Blending reactivity and goal-directedness in a fuzzy controller. En *Proceedings of the Second International Conference on Fuzzy Systems*, páginas 134–139, San Francisco, CA (USA), 1993.

[Saffiotti *et al.*, 1995] Alessandro Saffiotti, Kurt Konolige, y Enrico Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.

[Sahota *et al.*, 1995] Michael K. Sahota, Alan K. Mackworth, Rod A. Barman, y Stewart J. Kingdon. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. En *IEEE International Conference on Systems, Man and Cybernetics*, páginas 3690–3693, 1995.

[Serradilla and Maravall, 1996] Francisco Serradilla y Dario Maravall. A navigational system for mobile robots using visual feedback and artificial potential fields. páginas 1159–1164. Proceedings of the Thirdteenth European Meeting on Cybernetics and Systems Research, 1996.

[Serradilla and Maravall, 1998] Francisco Serradilla y Dario Maravall. A novel approach to autonomous navigation: the sensory gradient concept and its application to mobile robots. En *Proceedings of the Third IFAC Symposium on Intelligent Autonomous Vehicles*, páginas 383–388, Madrid, Spain, March 1998.

[Simmons and Mitchell, 1989] Reid Simmons y Tom M. Mitchell. A task control architecture for mobile robots. En *Working Notes of the AAAI Spring Symposium on Robot Navigation*, 1989.

[Simmons *et al.*, 1997] Reid Simmons, Richard Goodwin, Karen Zita, Sven Koening, y Joseph O'Sullivan. A layered architecture for office delivery robots. En Lewis W. Johnson, editor, *Proceedings of First International Conference on Autonomous Agents*, páginas 245–252, Marina del Rey, California (USA), 1997. ACM Press.

[Simmons, 1992] Reid Simmons. Concurrent planning and execution for autonomous robots. *IEEE Control Systems*, 12(1):46–50, 1992.

[Smith, 1980] Reid. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[Sommaruga and Catenazzi, 1996] Lorenzo Sommaruga y Nadia Catenazzi. *Lectures Notes on Artificial Intelligence*, capítulo From Practice to Theory in Designing Autonomous Agents, páginas 130–143. LNAI-1087. Springer-Verlag, 1996.

[Sommaruga *et al.*, 1996] Lorenzo Sommaruga, Ignacio Merino, Vicente Matellán, y José Manuel Molina. A distributed simulator for intelligent autonomous robots. En *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, páginas 393–399, Lisbon, (Portugal), 1996.

[Sommaruga, 1993] Lorenzo Sommaruga. *Cooperative heuristics for autonomous agents: an artificial intelligence perspective.* Tesis Doctoral, University of Nottingham, 1993.

[Steels, 1990] Luc Steels. Cooperation between distributed agents through self-organization. En Yves Demazeau y Jean-Pierre Muller, editor, *Descentralized A.I.* Elsevier Science, 1990.

[Stilwell and Bay, 1993] Daniel Stilwell y John Bay. Towards the development of a material transport system using swarms of ant-like robots. En *Proceedings of IEEE International Conference on Robotics and Automation*, páginas 766–771, 1993.

[Stone and Veloso, 1995] Peter Stone y Manuela Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. Informe técnico, School of Computer Science. Carnegie Mellon University, 1995.

[Stone and Veloso, 1997a] Peter Stone y Manuela Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. *International Journal of Human-Computer Systmes (IJHC)*, 1997.

[Stone and Veloso, 1997b] Peter Stone y Manuela Veloso. Using decision tree confidence factors for multiagent control. En *Workshop on RoboCup. Fifteen International Joint Conference on Artificial Intelligence IJCAI-97*, Nagoya, Japan, Agosto 1997.

[Stone and Veloso, 1998] Peter Stone y Manuela Veloso. Communication in domains with unreliable, single-channel, low-bandwidth communication. En *ICMAS-98 (submitted)*, Paris, Francia, Julio 1998.

[Tate *et al.*, 1994] Austin Tate, Brian Drabble, y Richard Kirby. *Intelligent Scheduling*, capítulo O-Plan2: an Open Architecture for Command, Planning and Control. Morgan Kaufmann, San Francisco (U.S.A.), 1994.

[Theraulaz *et al.*, 1990] Guy Theraulaz, Simon Goss, Jacques Gervet, y Jean-Louis Deneubourg. Task differentiation in polistes wasp colonies: a model for self-organizing groups of robots. En *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, páginas 346–355, 1990.

[Thrift, 1993] Philip Thrift. Fuzzy logic synthesis with genetic algorithms. En *Proceedings of the Third International Conference on Genetic Algorithms*, páginas 509–513, 1993.

[Thrun *et al.*, 1997] Sebastian Thrun, Arno Bücken, Wolfram Burgard, Dieter Fox, Thorsten Frölinghaus, Daniel Hennig, Thomas Hofmann, Michael Krell, y Timo Schmidt. Map learning and high-speed navigation in RHINO. En D Kortenkamp, Bonasso R.P., y Murphy R.R., editores, *AI-based Mobile Robots: Case studies of successful robot systems.* MIT Press, 1997.

[Thrun, 1996] Sebastian Thrun. A bayesian approach to landmark discovery and active perception for mobile robot navigation. Informe Técnico CMU-CS-96-122, Carnegie Mellon, School of Computer Science, Pittsburgh, PA 15213, Abril 1996.

[Valavanis and Saridis, 1992] Kimon P. Valavanis y George N. Saridis. *Intelligent Robotic Systems: Theory, Design and Applications.* Engineering and Computer Science. Kluwer Academic Publishers, 1992.

[Veloso *et al.*, 1995] Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, y Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, páginas 81–120, 1995.

[Walter, 1953] Grey Walter. *The Living Brain.* Duckworth, 1953.

[Weitzenfeld *et al.*, 1997] Alfredo Weitzenfeld, Ronald C. Arkin, Francisco Cervantes, Roberto Olivares, y Fernando Corbado. A neural schema architecture for autonomous robots. En *Conferencia Iberoamericana de Inteligencia Artificial*, 1997.

[Wiener, 1948] Norbert Wiener. *Cybernetics: Control and communication in the animal and the machine.* John Wiley, 1948.

[Zadeh, 1973] Lotfi A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, 1(1), 1973.

[Zadeh, 1988] Lotfi A. Zadeh. Fuzzy logic. *IEEE Transactions on Computers*, Abril 1988.

Índice de Materias